Universidad
Carlos III de Madrid
www.uc3m.es

Grado Universitario en Ingeniería
Electrónica Industrial y Automática
2015-2016

Trabajo Fin de Grado

"Estabilización de trayectorias mediante procesado multisensor"

Jorge Sánchez García

Director: Jesús García Herrero

Tutor: Abdulla Hussein Abdulrahman Al-Kaff

Universidad Carlos III de Madrid
Escuela Politécnica Superior
Avda. de la Universidad, 30 28911 Leganés, España

Octubre de 2016

Agradecimientos

Cada vez que llega el momento de concluir un nuevo reto académico, no puedo evitar recordar a todos los profesores que con su sabiduría, han ido moldeando la persona que hoy soy. A todos ellos les dedico este trabajo, y en especial a los profesores del Grado de Electrónica de la Universidad Carlos III, que me han acompañado en esta última etapa.

Quiero darle también las gracias a Jesús García, director de este trabajo, por sus ánimos para concluirlo.

Y por supuesto, quiero dedicarle este trabajo a mi familia y amigos por creer siempre en mí y porque con su apoyo es más fácil mirar hacia adelante.

"Estabilización de trayectorias mediante procesado multisensor"

Autor: Jorge Sánchez García

Director: Jesús García Herrero

Tutor: Abdulla Hussein Abdulrahman Al-Kaff

Universidad Carlos III de Madrid
Escuela Politécnica Superior
Avda. de la Universidad, 30 28911 Leganés, España

Octubre 2016

ÍNDICE

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

CAPÍTULO 1: INTRODUCCIÓN

- ***Presentación***
- ***Motivación personal***
- ***Objetivos***
- ***Contenido de la memoria***

PRESENTACIÓN

El presente trabajo consiste en la elaboración de un algoritmo de control en bucle cerrado que permita a un vehículo aéreo no tripulado seguir una trayectoria estable.

El UAV (por sus siglas en inglés) utilizado es un drone cuatricóptero elaborado por Parrot, modelo ARDrone 2.0. Con el fin de desarrollar aplicaciones para sus equipos, Parrot ofrece un SDK en el cual se encuentra la aplicación ARDRONE.CONTROL.NET desarrollada por Thomas Endres, Stephen Hobley y Julien Vinel en Visual C# 2010 y que cuenta con un completo juego de librerías para actuar sobre el drone. El Laboratorio de Sistemas Inteligentes de la Universidad Carlos III, actualiza y añade funcionalidades como la detección del movimiento por visión artificial, de la cual nos valdremos para analizar el movimiento del drone, renombrando el proyecto como LSIDrone. Francisco Javier García Muñoz, alumnos de Ingeniería Electrónica Industrial en la Universidad Carlos III, toma como base este proyecto para realizar su Trabajo Fin de Grado.

Este trabajo es continuación directa del Trabajo Fin de Grado desarrollado por Francisco Javier García Muñoz "Técnicas de Control de Vuelo sobre un Cuadricóptero" entregado en Febrero de 2015. En dicho trabajo Francisco Javier expone un algoritmo de control en lazo abierto que permite al drone seguir trayectorias en un plano horizontal.

Valiéndonos de los recursos disponibles, como son la información de los sensores, la detección de movimiento desarrollada por Abdulla Al-Kaff, profesor y miembro del Laboratorio de Sistemas Inteligentes de la Universidad Carlos III, y las mejoras implementadas por Francisco Javier García Muñoz, es objeto la realización de un algoritmo de control en bucle cerrado que dote de movimiento y orientación al drone en un espacio tridimensional.

Este escrito detalla los procedimientos seguidos.

MOTIVACIÓN PERSONAL

La motivación fundamental para realizar este trabajo es el uso de técnicas de control, ya que al término del grado me he ido especializando en materias enfocadas a la electrónica, obviando la parte de control y automática que también compone el Grado de Ingeniería Electrónica y Automática.

Por supuesto también despierta mi interés el hecho de trabajar con drones. Aunque los drones de cierta envergadura dejan de ser simples juguetes, este hecho no les resta diversión. Y aunque me parece que el mercado empieza a estar algo saturado de los drones clásicos, que en esencia son todos iguales, si creo que queda hueco para la alternativa.

Otro aspecto que me llamó la atención fue el uso del entorno de desarrollo Visual Studio, y más en concreto, el lenguaje Visual C#. En cursos precedentes de la citada titulación he podido disfrutar (y también sufrir) con la potencia de lenguajes como C y C++, y tenía curiosidad por conocer las ventajas que ofrece su evolución C# (sin olvidar la parte de sufrir).

OBJETIVOS

El principal objetivo es el desarrollo de un algoritmo de control, que le permita a un usuario comandar los movimientos de un drone, simplemente especificando las trayectorias a seguir por medio de puntos en un espacio tridimensional.

Objetivos del proyecto

1) Obtener lecturas del sensor inercial del drone y/o del movimiento a través de la visión artificial
2) Conseguir un control estático manteniendo el drone en un punto fijo conocida su posición tridimensional y la orientación.
3) Seguir trayectorias rectas desde un punto origen a otro punto destino
4) Concatenar trayectorias previamente introducidas en una interface gráfica, para realizar recorridos complejos

Objetivos personales

1) Poner en práctica las técnicas de control aprendidas durante la carrera en una aplicación real.
2) Iniciarme en el uso de C# como lenguaje de programación.

CONTENIDO DE LA MEMORIA

Esta memoria se divide en ocho capítulos:

1) Introducción

   Presentación de la problemática y los objetivos a cumplir.

2) Estado del arte

   Puesta en situación del contexto actual

3) Planteamiento general

   Recursos disponibles y solución propuesta

4) Desarrollo

   Descripción de la solución propuesta y su consecución

5) Pruebas y resultados

   Validación de las tesis planteadas

6) Conclusiones y trabajos futuros

   Síntesis de los resultados obtenidos y proposición de mejoras

7) Gestión del proyecto

   Análisis de los recursos empleados, humanos y materiales

8) Documentación de consulta

   Relación de documentos consultados para la consecución de este trabajo

Y dos anexos:

A. Definición de tipos

   Listado exhaustivo de los tipos de datos implementados en este trabajo

B. Cálculos de navegación

   Exposición de cálculos realizados para el algoritmo de navegación

CAPÍTULO 2:
ESTADO DEL ARTE

- ***Drones. Aspectos generales***
- ***Modelo ARDrone 2.0 Parrot***
- ***Otros modelos***
- ***Sensores***

DRONES. ASPECTOS GENERALES

La definición de "dron" según la Real Academia de la Lengua Española es, *"aeronave no tripulada"*.

Otra definición más común es *"Vehículo Aéreo no Tripulado"* que da origen a sus siglas VANT (UAV, *"Unmanned Aerial Vehicle"*, en inglés). [1]

Como su nombre indica, el drone consiste en una aeronave que no requiere de un piloto en su interior para ser manejado. Su control se realiza tanto en remoto como en vuelo autónomo. Esto los hacen ideales para:

- Aplicación peligrosas, como actuaciones en conflictos armados o control de incendios
- Inserciones en entornos agresivos para el hombre como zonas de alta radioactividad o cráteres de volcanes activos para su estudio.
- Largos periodos de operación, como vigilancia y control del terreno
- Aplicaciones de tamaño reducido como seguimiento de eventos deportivos, donde las aeronaves convencionales no tienen cabida.
- Aplicaciones recreativas, entre otras.

Pueden clasificarse siguiendo varios criterios: [2]

- Tipo de misión
  - Reconocimiento u observación
  - Investigación
  - Salvamento
  - Anti-incendios
  - Combate
  - Transporte
  - Blancos aéreos
- Origen de misión
  - Civil
  - Militar
- Tamaño
  - Grande
  - Mediano
  - Pequeño
  - Micro UAV

- Forma de sustentación
  - Más pesados que el aire
  - Más ligeros que el aire
  - Híbridos
- Forma de despegue
  - Desde pista
  - Lanzados a mano
  - Lanzados con medios mecánicos
- Tipo de motor
  - Alternativo
  - Turbina
  - Eléctrico
- Duración de la misión
  - Larga duración
  - Media duración
  - Corta duración
- Cota de vuelo
  - Alta o muy alta cota
  - Media cota
  - Baja cota
- Tipo de Control
  - Autónomo o adaptativo
  - Monitorizado
  - Supervisado
  - Autónomo no adaptativo o pre-programado
  - Dirigido por un operados

Durante los últimos años, se ha popularizado el modelo multi-rotor, especialmente en aplicaciones civiles de tipo industrial o recreativo, como control de cosechas o eventos deportivos. Esto es debido a su versatilidad, despegue vertical que facilita su operación, estabilidad y sustentación en un punto fijo.

Por el contrario, su tiempo de vuelo es menor, ya que toda la sustentación recae sobre el empuje vertical de los motores, lo que multiplica el consumo.

Existen varias configuraciones de cuatro, seis, ocho o incluso más rotores, pero siempre en número par. El motivo de ampliar el número de rotores es que al aumentar su número, se mejora la estabilidad

MODELO ARDRONE 2.0 PARROT

Modelo utilizado para el desarrollo de este trabajo. Es un cuadricoptero de motores eléctricos alimentados por batería de LiPo, destinado a uso recreativo o académico.

Asistencia electrónica: [3]

- Procesador: Procesador 1 GHz 32 bits ARM Cortex A8 con DSP vídeo 800 MHz TMS320DMC64x
- Sistema operativo: Linux 2.6.32
- RAM: DDR2 1 GB a 200 MHz
- USB: USB 2.0 de alta velocidad para las extensiones
- Wi-Fi: Wi-Fi b g n
- Giroscopio: 3 ejes, precisión de 2000°/segundo
- Acelerómetro: 3 ejes, precisión de ± 50 mg
- Magnetómetro: 3 ejes, precisión de 6°
- Sensor de presión: Precisión de ± 10 Pa
- Sensores de ultrasonidos para medir la altitud: Medición de la altitud
- Cámara vertical: QVGA 60 FPS para medir la velocidad en vuelo

Grabación de video:

- Cámara HD: Cámara HD 720p 30 FPS
- Objetivo: Objetivo gran angular: diagonal 92°
- Perfil de codificación básica: H264
- Formato fotos: JPEG
- Conexión: Wi-Fi

Motorización:

- 4 motores sin escobillas de tipo "inrunner": 14,5 vatios y 28 500 rpm
- Rodamiento de bolas en miniatura

- Engranajes Nylatron
- Rodamiento de bolas autolubricante de bronce

Peso:

- Con carena interior: 380 g
- Con carena exterior: 420 g

Ilustración 1.- Imagen ARDrone 2.0 Elite Edition Snow

El nuevo modelo ARDrone 2.0, mejora notablemente a su predecesor, sin embargo, las librerías de que disponemos en C# para comunicarnos y controla el drone, fueron escritas para la primera versión, por lo que la mayoría de las funcionalidades nueva en el modelo 2.0, no está disponibles, como por ejemplo, el magnetómetro de 3 ejes.

OTROS MODELOS

La misma empresa Parrot dispone de más modelos de drones como el Bebop 2 o el Disco, que ofrecen una experiencia inmersiva y largo alcance para realizar vuelos complejos. Además de otros modelos de menor tamaño.

Ilustración 2.- Distintos modelos de Parrot
En la parte superior se encuentra el modelo Bebop 2, y en la inferior, el modelo Disco

Otro competidor es Walkera, que cuenta con drones aéreos para vuelo sostenido y grabación de imágenes, y de carreras, para competición.

Ilustración 3.- Distintos modelos de Walkera
En la parte superior se encuentran los modelos aéreos, y en la inferior, los de carreras

Otra alternativa es fabricarse uno mismo su propio drone.

En este sentido hay varias opciones, como por ejemplo Pixhawk, que sigue la filosofía de hardware libre. [4]

- 32bit STM32F427 Cortex M4 core with FPU
- 168 MHz
- 256 KB RAM
- 2 MB Flash
- 32 bit STM32F103 failsafe co-processor

Ilustración 4.- Placa de desarrollo Pixhawk

SENSORES

Los sensores son imprescindibles para la estabilización de las trayectorias, ya que nos permiten muestrear los estados del sistema y realimentar el control.

Los principales sensores que podemos encontrar en un dron para este fin son:

- Acelerómetros: Miden aceleración en cualquier eje. De este modo también pueden sensar la horizontalidad midiendo la aceleración de la gravedad. Integrando su señal de salida se puede obtener la velocidad y la posición.
- Giróscopos: Miden la orientación en cualquier eje.
- Magnetómetro: Miden la orientación respecto de los polos magnéticos. Muy útiles para navegación
- Ultrasonidos: Miden la distancia al siguiente objeto mediante el eco de una onda reflejada.

Otros sensores comunes son cámaras de video, GPS y sensores de presión para medir la altura barométrica.

El modelo ARDrone 2.0 cuenta con todos ellos, salvo el GPS, que se instala mediante accesoria aparte. Sin embargo, más adelante comprobaremos que la información aportada por estos sensores es de mala calidad y difícil de tratar.

CAPÍTULO 3: PLANTEAMIENTO GENERAL

- ***Problema propuesto***
- ***Recursos disponibles***
- ***Algoritmos de control***
- ***Solución adoptada***

PROBLEMA PROPUESTO

En el primer capítulo de este trabajo, se ha introducido la problemática a salvar.

Con una visión más en concreto, este trabajo trata de continuar con el trabajo elaborado por Francisco Javier García Muñoz. Francisco Javier formuló un algoritmo que controlaba el drone en lazo abierto, lo que propiciaba trayectorias erráticas, como era de esperar, ya que el drone no puede responder ante acciones externas como ligeras brisas, ni internas como ligeras variaciones en la velocidad de los rotores, que en suma, van perturbando el trayecto esperado.

Por ello se propone la elaboración de un nuevo algoritmo de control, esta vez en lazo cerrado. El algoritmo debe realizar eficazmente un control sobre la posición en un espacio tridimensional, y la orientación. Y por medio de este control, ser capaz de desplazarse de manera fiel a lo largo de una serie de trayectorias indicadas por el usuario a través de un archivo de texto o cualquier otro método.

RECURSOS DISPONIBLES

Compendio de herramienta, documentos y similares disponibles para resolver la problemática propuesta.

Aplicación LSIDrone

El punto de partida es esta aplicación creada por el Laboratorio de Sistema Inteligentes de la Universidad Carlos III, y mejorada con nuevas funcionalidades por Francisco Javier García Muñoz.

Ilustración 5.- Interfaz LSIDrone

Esta aplicación sirve de propósito para distintas investigaciones dentro del Laboratorio de Sistemas Inteligentes, sin embargo solo nos centraremos en las funcionalidades de conexión con el drone y monitorización de su estado, control manual e instrumentación.

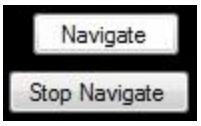
- En la parte superior, podemos encontrar los botones de “Conexión” para enlazar nuestro ordenador con el drone, “Desconexión” para realizar la operación contraria, “Flat Trim” para ajustar el offset de los sensores a cero y renivelar el drone, y “Emergencia” para apagar súbitamente el drone en caso de necesidad.

- También en la parte superior encontramos informaciónsobre el estado del drone, como el nivel de batería, si está conectado o no, si el sistema detecta que está en vuelo, si se mantiene suspensión en el áire, o si se ha activado el estado de emergencia.

- En la zona central se hubican los instrumentos de vuelo, que ofrecen información acerca del movimiento del vehículo, como la orientación , giro o velocidad ente otros.

- Situado en la parte derecha, hayamos los botones de control manual para elevar, mover o rotar el drone entre otros movimientos.

- Justo encima de los botones de control manual, podemos encontrar los botones "Navigate" y "Stop Navigate", que junto con el botón "Change Camera" en la zona izquierda la pantalla, nos sirven para activar la detección de movimiento por visión artificial.



- Además de distintas etiquetas que nos ofrecen información de los sensores o datos procesados a cerca de la velocidad y posición, ente otros.

Nos valdremos de esta aplicación para probar el comportamiento del drone y monitorizar su estado.

Código aplicación LSIDrone

Profundizando en la aplicación, nos encontramos el código, escrito en C#.

Está organizado en una solución llamada LSIDrone, que se compone de diez proyectos, uno ejecutable, y los otros nueve, librerías *dll*:

- LSIDroneInterface
- GraphLib
- LSIDroneAviationInstruments
- LSIDroneBasics
- LSIDroneCapture
- LSIDroneControlLibrary
- LSIDroneDetection
- LSIDroneHudInstruments
- LSIDroneInput
- LSIDroneTesting

El proyecto ejecutable es “LSIDroneInterface”, y los otros proyectos son librerías, en gran parte, herencia directa de la aplicación ARDRONE.CONTROL.NET desarrollada por Thomas Endres, Stephen Hobley y Julien Vinel.

El proyecto “LSIDroneInterface” se compone de diferentes formularios, clases y recursos. El formulario más importante es “MainForm.cs” que implementa la clase “MainForm”, la cual describe todos los componentes de la interfaz gráfica principal y el código que los gobierna.

Esta clase contiene numerosos miembros y métodos. Se encuentran entre los más destacables:

- *DroneData data* Sirve de contenedor para el volcado de los datos provenientes de la IMU. Accediendo a sus miembros, podemos obtener valores de velocidad, altura, rotación, etc.
- *private DroneControl droneControl* Nos permite enviar comandos al drone, y leer de sus sensores información que luego volcaremos en la variable *data*.

- *private void UpdateStatus()* Este método se ejecuta periódicamente por acción de un Timer y realiza distintas funciones, como leer valores de la IMU o actualizar la interfaz gráfica. Pero su función más destacable es ejecutar el algoritmo de detección de movimiento que le permite al drone conocer su posición y orientación por medio de la visión artificial.
- *Private double Xacc, Yacc, Zacc, Acumtheta* Estos miembros almacenan la posición y orientación resultante del algoritmo de detección de movimiento,

Esta clase, también contiene el código desarrollado por Francisco Javier durante su TFG, además de numerosos métodos para comandar el vehículo de Parrot.

En cuanto a las librerías, la más útil para nuestro propósito es “LSIDroneControlLibrary”, que incluye las clases “DroneData” y “DroneControl”, las cuales implementan la capacidad de comandar el drone, o leer su estado.

SDK de Parrot

En el enlace web https://github.com/Parrot-Developers, tenemos a nuestra disposición toda la documentación referente al desarrollo de aplicaciones para los productos Parror, y más en concreto para el drone ARDrone 2.0, del cual hacemos uso.

En la mencionada documentación se encuentran manuales de usuario y desarrollador, entornos de desarrollo con ejemplos en distintas plataformas y la aplicación ARDRONE.CONTROL.NET

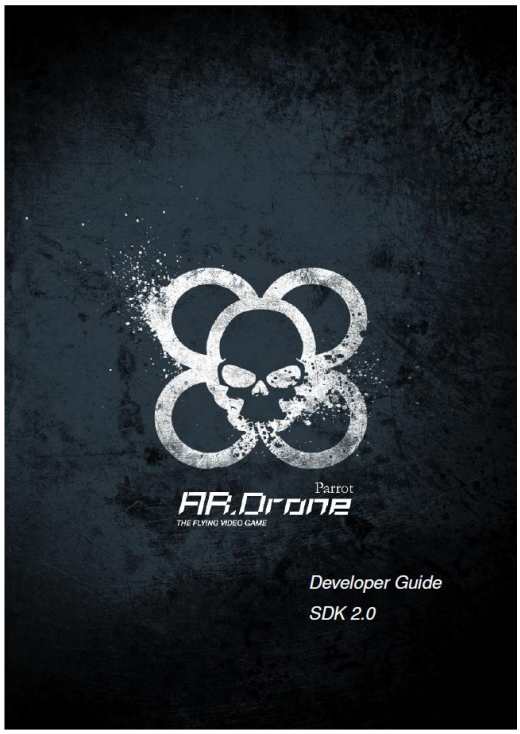


Ilustración 6.- Portada manual para desarrolladores

Ilustración 7.- Interfaz ARDrone

ALGORITMOS DE CONTROL

Para llevar a cabo el control sobre el drone, se van a implementado dos algoritmos:

El primero representa el núcleo del control, puesto que ejerce la acción de control sobre la posición y la velocidad del vehículo. Se puede considerar un control ESTÁTICO, desde el punto de vista de que el drone intentará mantener una posición previamente establecida. De esta manera, el drone intenta corregir el error de posición producido entre su posición actual, y la entrada que representa la posición de consigna, y al mismo tiempo controla la tasa con la que se reduce este error de posición, es decir, la velocidad.

El segundo algoritmo consiste en una extensión del primero. Si en el primer algoritmo el error de posición se establece entre la posición actual y el destino deseado. Este segundo algoritmo traza una trayectoria recta entre el punto de partida y el destino, siendo el error de posición la distancia entre el drone y la línea recta que representa esta trayectoria ideal. El error de posición se traduce en un vector velocidad que intenta corregirlo, y a este vector velocidad se le suma otro vector velocidad con la misma directriz que la línea que une el punto de partida y el destino, para favorecer el movimiento de la aeronave a lo largo de la trayectoria. Es por esto un control DINÁMICO sobre la trayectoria.

Algoritmo de Control Estático

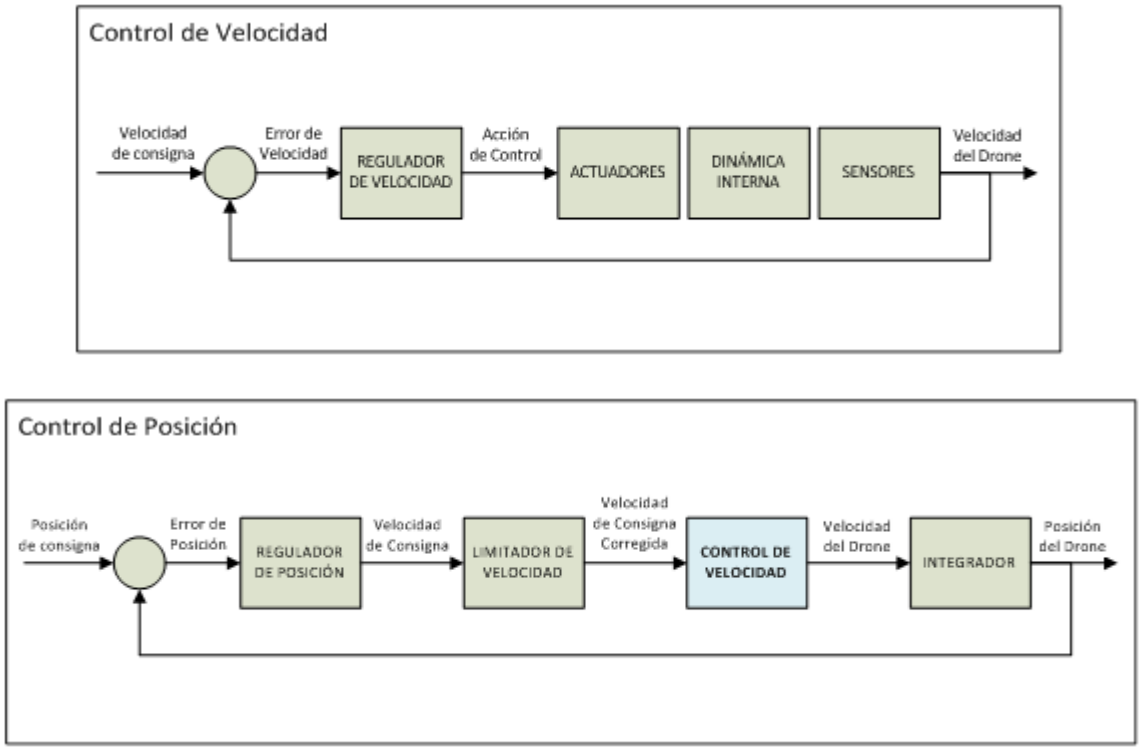


Ilustración 8.- Esquema del Algoritmo de Control Estático

Como se ha introducido en el apartado anterior, el algoritmo obtiene la velocidad y la posición actual por medio de los sensores inerciales de que dispone el drone, y compara las medias con la velocidad y posición deseadas. Algunas magnitudes se obtienen de la IMU en forma de velocidad, como son la velocidad en los ejes X e Y. Mientras que otras magnitudes tienen carácter posicional, como la altura y los ángulos de rotación.

Integraremos las velocidades para calcular la posición, y derivaremos la altura y los ángulos para conocer su tasa de variación. En el capítulo de *Desarrollo* trataremos con detalle cómo están construidas las clases que implementan el cálculo de la integración y la derivación.

Una vez conocidos los estados del sistema, se llevará a cabo la acción de control con la secuencia siguiente:

1) **Cálculo del error de posición.** El error de posición representa la diferencia entre la posición deseada o de consigna, y la posición real en la que se encuentra el vehículo.

2) **Regulador de posición.** Se ha implementado un regulador proporcional consistente en una constante Kp que multiplica al error. Como se detallará más adelante en el capítulo de *Desarrollo*, la clase que implementa al controlador ha sido escrita como un PID con constantes Kp, Ki, y Kd, programables según se necesite, para permitir una funcionalidad completa. Sin embargo, para regular la posición solo se ha tenido en consideración la versión más simple del regulador, el regulador proporcional. Esto de debido a que la salida del regulador representa la velocidad de consigna como un vector con la misma directriz que la recta que une la posición actual con el destino. Un regulador PI o PID podría desacompasar la velocidad en un eje con respecto a los otros dos, y la trayectoria tendería a alejarse de una línea recta.

3) **Limitador de velocidad.** La expectativa de realizar una acción en el menor tiempo posible, conlleva una tasa de variación infinita, lo cual es imposible y su intento solo conllevaría dificultad en el control. Por ello se limita la velocidad a un valor preestablecido por el usuario.

4) **Cálculo del error de velocidad.** De manera similar al planteamiento para la posición, se obtiene el error de velocidad, como la diferencia entra la velocidad deseada o de consigna, y la velocidad actual del drone.

5) **Regulador de velocidad.** En esta ocasión se ha optado por un regulador proporcional-integral. Se ha incluido la parte integral para que el regulado tenga memoria, y pueda corregir los efectos de acciones permanentes como una ligera brisa, o en especial, el rozamiento con el aire.

6) **Actuadores.** La tentativa de acción sobre un eje, se traduce en una orden hacia los motores.

Nota.- *Debido a que no se conoce la dinámica interna del drone, los parámetros de los reguladores han de calcularse experimentalmente, e introducirse en la aplicación gráfica.*

Algoritmo de Control Dinámico

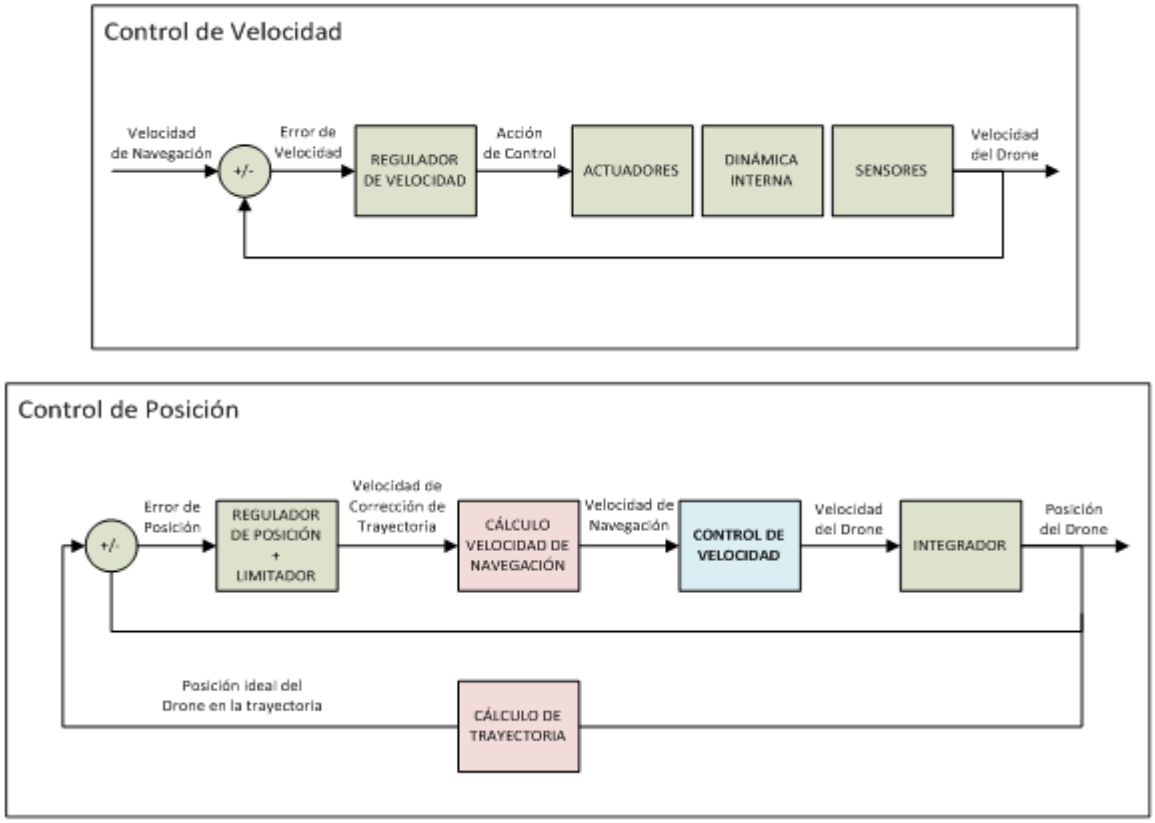


Ilustración 9.- Esquema del Algoritmo de Control Dinámico.

Este segundo algoritmo es una extensión del primero, al cual se le han añadido las funcionalidades de los bloques tintados en rojo de la ilustración superior.

Como se ha descrito anteriormente, este algoritmo difiere del primero en que el control de posición se realiza sobre una línea recta que representa la trayectoria a seguir, y se desarrolla desde el punto inicial del movimiento u origen, hasta el punto final o destino.

Para ello, el drone compara su posición con la que debería ocupar en la trayectoria, y de este modo se genera el error de posición que desencadenará la acción de control haciendo uso del motor de control descrito en el primer algoritmo. Este motor calcula la velocidad aplicable necesaria para corregir su posición respecto de la trayectoria deseada.

Hasta este instante, hemos conseguido que el drone corrija su posición con respecto a la trayectoria, pero no que la recorra. Para recorrer la trayectoria y forzar el movimiento desde el origen hasta el punto de destino, a la velocidad anteriormente calculada, se le suma otro vector

velocidad con la misma directriz que la trayectoria ideal, siendo el resultado la acción combinada de corrección de posición y recorrido de la trayectoria.

- **Cálculo de Trayectoria.** Conocido el punto A (origen), el punto B (destino), y el punto P (posición actual del drone), se calcula el punto P' como el punto perteneciente a AB más cercano a P, siendo PP' el error de posición.
- **Cálculo Velocidad de Navegación.** Según se ha comentado en el párrafo anterior, es necesario realizar una suma vectorial de velocidades para forzar el movimiento a lo largo de la trayectoria. Para ello, a la velocidad resultante del control de posición, se le suma un vector velocidad con la misma directriz que el vector AB. El cálculo se realiza tal que la combinación de vectores dé como resultado un vector velocidad cuyo módulo sea igual a la velocidad máxima permitida por el usuario.

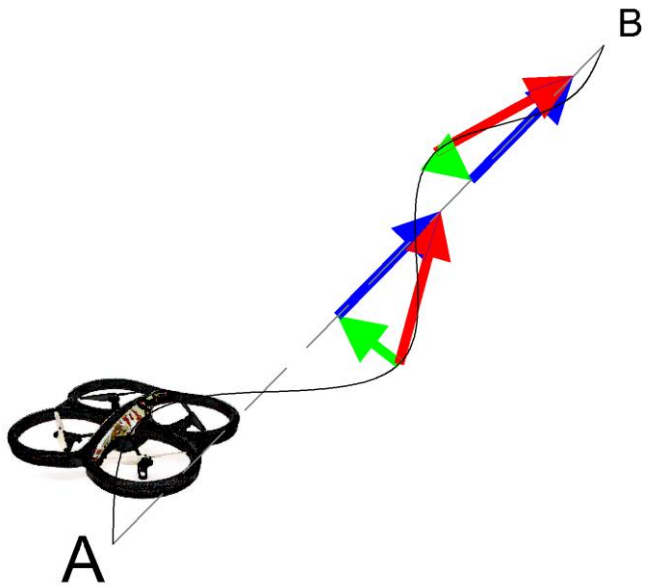


Ilustración 10.- Velocidad de Navegación.
En verde, velocidad de corrección de la posición. En azul, velocidad de recorrido de trayectoria. Y en rojo, velocidad resultante de la combinación de ambos vectores.

SOLUCIÓN ADOPTADA

Para implementar estos algoritmos, se ha creado una nueva librería de clases que contiene el código que los describe, y un nuevo formulario para permitirle al usuario valerse de los recursos de esta librería de clases.

Nueva librería de clases

Para construir el control, se ha añadido un nuevo proyecto de librería a la solución, con el nombre "LSIDroneFlightControl", el cual contiene las clases que servirán para implementar el algoritmo de control.

- **clsSystemInfo**

  Clase base para recopila la información básica para el control, como el estado, o la posición en un sistema cartesiano absoluto. El algoritmo de control debe ser independiente de la fuente de datos. Con este fin, esta clase debe servir de base para clases derivadas según el método de obtención de datos, IMU, procesamiento de imágenes o GPS. Los atributos y métodos comunes como el timer se implementan en esta clase, mientras que los métodos que difieren según el caso, se definen como *virtual* para poder ser sobrecargados.

- **clsSystemInfo_IMU**

  Clase derivada de "clsSystemInfo" con los métodos sobrecargados para obtener información de la unidad inercial, procesarla y obtener la posición y velocidad respecto de un sistema cartesiano absoluto.

- **clsSystemInfoIMAGE**

  Clase derivada de "clsSystemInfo" con los métodos sobrecargados para obtener información de posición a través del procesamiento de las imágenes ofrecidas por la cámara, para después procesarla y obtener la posición y velocidad respecto de un sistema cartesiano absoluto.

- **clsControlSystem**

  Esta es la clase más importante, ya que representa el núcleo del control. En ella se codifica el Algoritmo de Control Estático, y servirá de clase base para la clase derivada que implemente el Algoritmo Dinámico. Así, el método que se encarga de

secuenciar el algoritmo de control, está definido como *virtual* para poder ser sobrecargado. Su atributo más destacable es un puntero tipo "clsSystemInfo" que será inicializado con una instancia del tipo "clsSystemInfo_IMU" o "clsSystemInfo_IMAGE", según proceda. A través de este puntero, se obtienen los valores de posición y velocidad.

- **clsNavigationSystem**

  Hereda de "clsControlSystem", y añada funcionalidades que le permiten realizar un control de posición y velocidad sobre una trayectoria ideal que une dos puntos en línea recta. Sobrecarga el método que secuencia el control, para poder introducir las variaciones de este algoritmo como son llamadas a los métodos que calculan la posición ideal dentro de la trayectoria y la velocidad necesaria para recorrerla.

- **clsCommands**

  Los métodos de esta clase son una colección de instrucciones para comandar el drone, tales como despegar, aterrizar, mantenerse volando en suspensión o ajuste de los giróscopos. Estos métodos son una recopilación de métodos presentes en la clase "MainForm", con algunas ligeras modificaciones para devolver cadenas de que incluyan un texto descriptivo de cada comando.

- **clsFlightPlan**

  Clase gestora de las trayectorias. Sus métodos permiten añadir y borrar puntos de una lista de objetos que se encuentra entre sus atributos, salvar y leer trayectorias en un archivo de texto plano, o devolver información sobre el siguiente punto en la trayectoria. Otro atributo destacable es un puntero de tipo "CheckedListBox", que le permite al desarrollador dibujar en el formulario un control del mismo tipo que el puntero, y vincularlo a esta clase, para poder mostrar las trayectorias por pantalla.

- **clsAutomaticPilot**

  Clase derivada de "clsCommands", hereda todos sus métodos. Sus atributos incluyen un tipo "clsControlSystem" que será instanciado con tipo "clsNavigationSystema" para sobrecargar los métodos referentes al control. Y otra instancia de tipo "clsFlightPlan". Esta clase sirve de capa de abstracción al usuario, encargándose de todos los aspectos referentes a la navegación y control del drone,

valiéndose de los miembros heredados de “clsCommands”, y de las funcionalidades de “clsNavigationSystem” y “clsFlightPlan”, a las que tiene acceso.

- **clsP_I_D_Regulator**

  Esta clase tiene todas las funcionaliadades necesarias para implementa un regulador PID. Sus tres constantes son ajustables, y la manear de activar las componentes integrales y derivativas es estableciéndoles un valor distinto de cero. Se ha usado la clase “clsIntegrator” para implementar la parte integral, y la clase “clsDifferentiator”, para la parte diferencial.

- **clsDifferentiator**

  Calcula la diferencia de una señal discreta comparando el valor actual con el último valor adquirido.

- **clsDifferentiatorAngle**

  Clase derivada de “clsDifferentiator”, su uso se basa en la necesidad de tratar de manera especial la derivación de la orientación. El problema surge cuando el drone, rotando sobre su propio eje, sobrepasa la posición de 360º, y la orientación salta bruscamente de 360º a 0º, produciéndose un error en el resultado de la derivación. Esta clase sobrecarga el método encargado de realizar el cálculo, y cuida de evitar este error.

- **clsIntegrator**

  Calcula la integral de la señal por el método trapezoidal, comparando el valor actual con el último valor adquirido y sumando el resultado al valor acumulado.

- **clsIntegratorAngle**

  Clase derivada de “clsIntegrator”, sobrecarga el método que realiza el cálculo para corregir el valor de salida, ya que el ángulo solo puede tomar valores entre $(-\pi, \pi]$.

- **clsFilter**

  Esta clase implementa un filtro promediador de tamaño de buffer ajustable en el momento de su instanciación.

- **clsPoint**

  Almacena un punto de la trayectoria a seguir.

Nuevo formulario

Este formulario le sirve al usuario para monitorizar el estado del drone y la acción de control, ajustar los parámetros de los reguladores, definir un destino, y para crear, guardar y cargar trayectorias que posteriormente el drone deberá realizar.

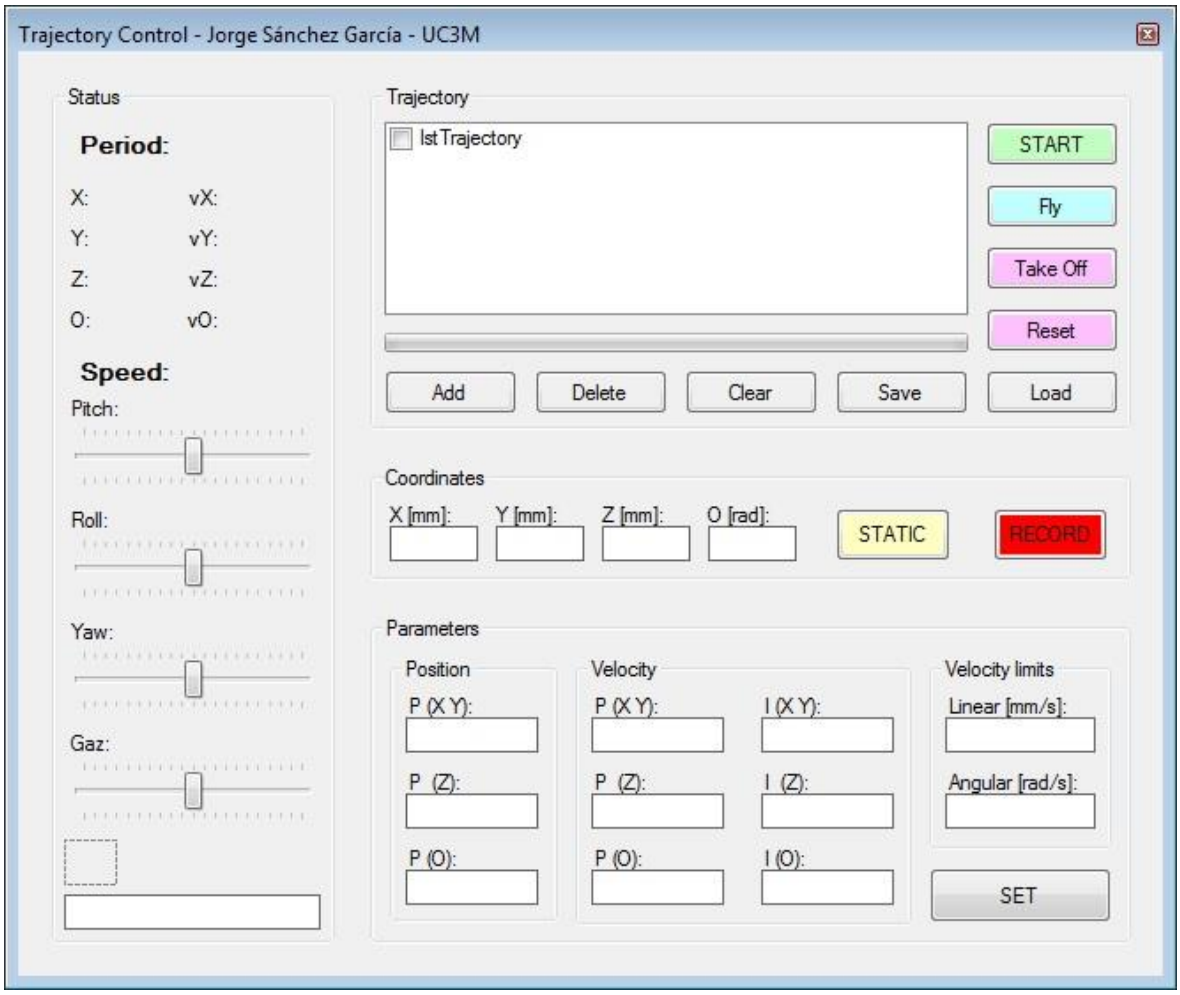


Ilustración 11.- Nuevo formulario para el control de trayectorias

- En la zona izquierda del formulario, se encuentra del grupo de controles “Status”, que nos permite monitorizar el estado de control del drone. Monitorizamos el periodo de muestreo, que nunca será constante, ya que el programa no corre en tiempo real. La posición y orientación del drone en una espacio cartesiano absoluto, la velocidad respecto de cada eje y el módulo de velocidad absoluta. La acción de control que el programa comanda a los actuadores con controles tipo slider para mostrar la intensidad de manera visual, y un cuadro de texto para mostrar información.

- En la parte superior se ubica el grupo “Trajectory”, con un listado de las trayectorias a ejecutar, controles para iniciar el control, una barra de progreso para indicar el progreso de cada trayectoria desde el origen hasta su destino, y controles para añadir y borrar trayectorias a la lista, y para guardar y cargar trayectorias en un archivo de texto plano.
- En el centro de formulario, el grupo de controles “Coordinates” nos permite introducir coordenadas, ordenar al drone a desplazarse a una punto concreto en el plano cartesiano, o activar la grabación de telemetría del vuelo.
- En la parte inferior, los controles del grupo “Parameters”, le permiten al usuario modificar los parámetros proporcional e integral de los reguladores de posición y velocidad, además de establecer la velocidad lineal y angular máxima para limitar la respuesta del drone y facilitar su control.

CAPÍTULO 4: DESARROLLO

- ***Mapa de clases***
- ***Diagramas funcionales***
- ***Funcionalidades del formulario***
- ***Llamadas a controles en hilo seguro***

MAPA DE CLASES

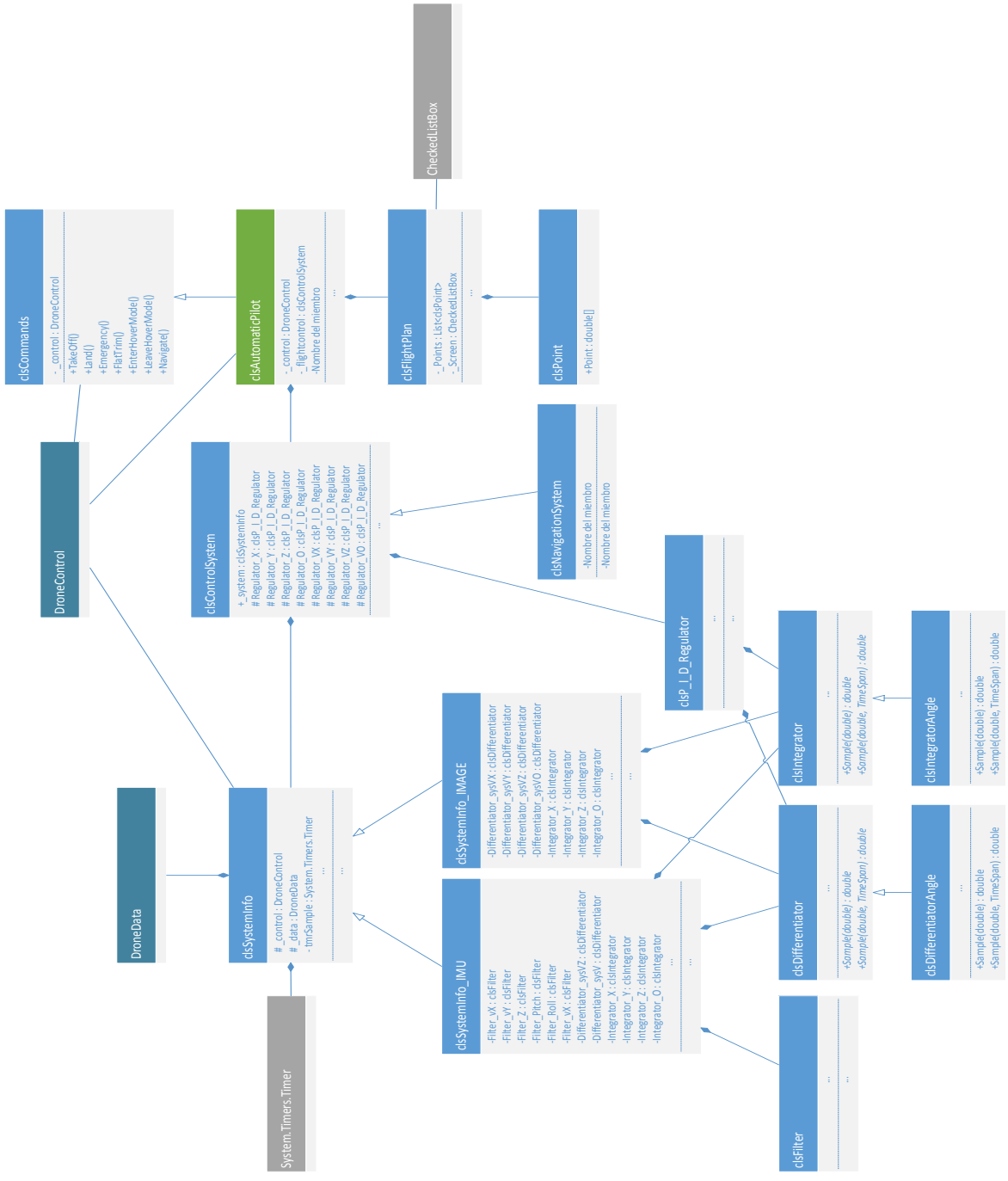


Ilustración 12.- Mapa de Clases

Nota.- Se puede encontrar el listado completo de miembros de las clases y sus funciones, en el Anexo A: Definiciones de tipos.

DIAGRAMAS FUNCIONALES

Los siguientes diagramas no pretenden describir el código de programación escrito en cada clase, sino la funcionalidad que realiza la correspondiente clase. Esto significa que un bloque del diagrama no tiene por qué corresponder necesariamente con un método de la clase en cuestión.

Visión general

Con este diagrama, al mapa de clases visto en el apartado anterior, se le aporta un camino de acciones secuenciales que clarifique sus cometidos.

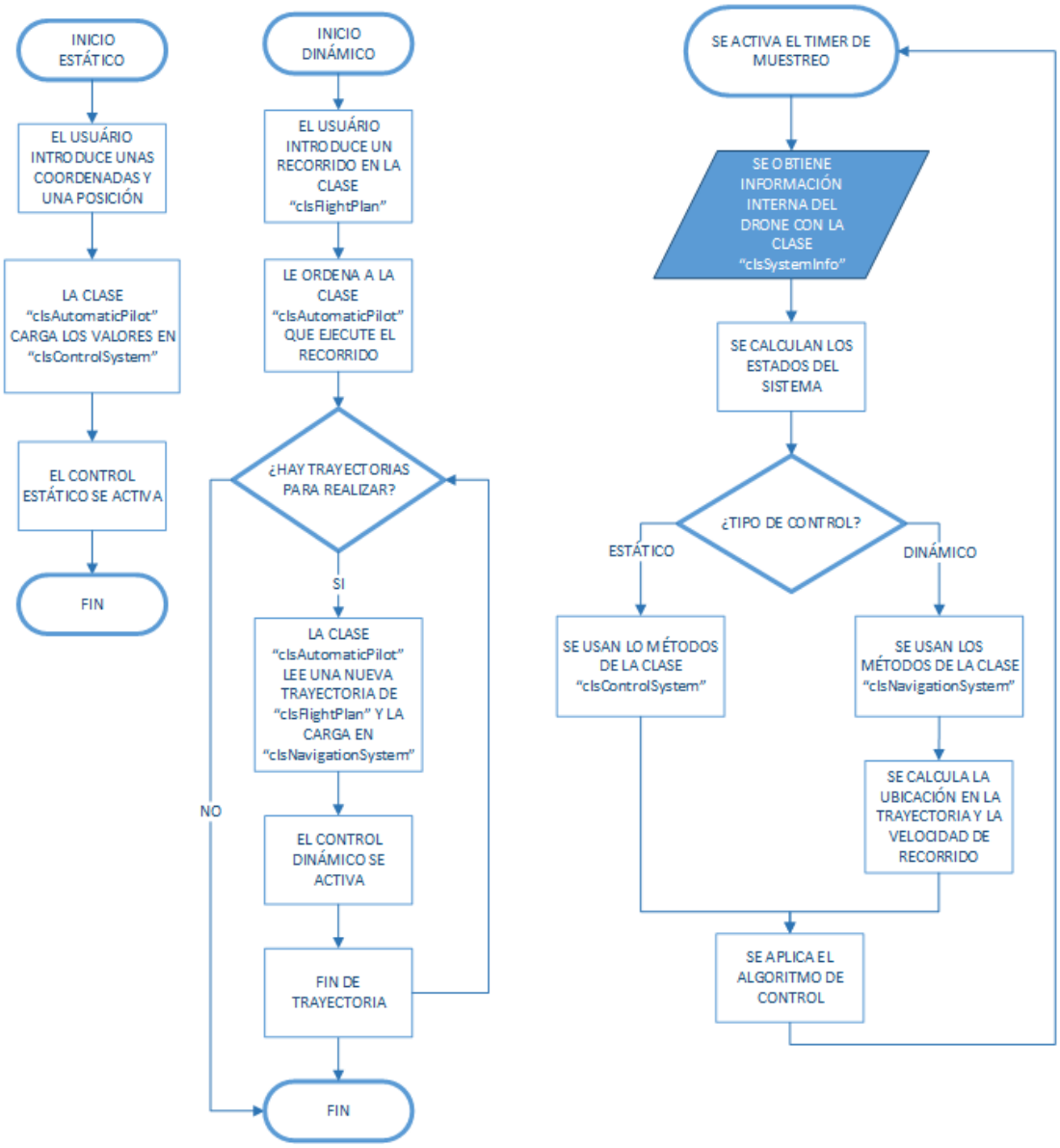


Ilustración 13.- Diagrama global funcional

Clase clsSystemInfo

Esta clase tiene un timer que periódicamente llama a la función delegada asociada. Está función llama al resto de miembros de la clase, desencadenándose así la acción de control. En esta clase no se implementa el algoritmo de control, solo sirve para recopilar información y conocer los estados del sistema. Por ello esta clase se incluye entre los atributos de clsControlSystem, que sí realiza el control. Para llamar al método que realiza el control, clsSystemInfo contiene miembro de tipo delegado que ha de ser asociado al método correspondiente de clsControlSystem, o de clsNavigationSystem, como veremos más adelante.

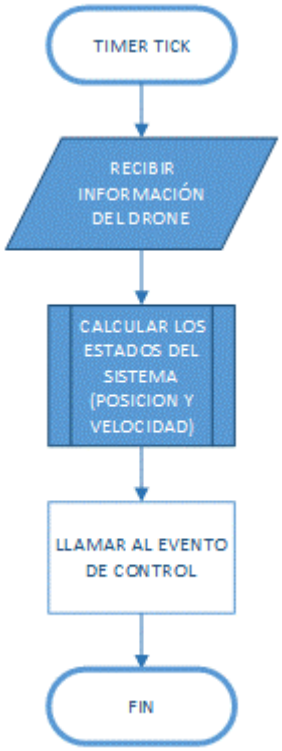


Ilustración 14.- Diagrama funcional clsSystemInfo

Clase clsSystemInfo_IMU

Esta clase es herencia de su clase padre clsSystemInfo, por lo tanto la funcionalidad es exactamente la misma. Solo varía en el cálculo de los estados del sistema, ya que en la clase clsSystemInfo, el método correspondiente al cálculo de estados esta vacio, y se ha definido como *virtual* para poder ser sobrecargado. Así se permite al algoritmo, usar diferentes fuentes de datos, simplemente creando otra clase derivada de clsSystemInfo y sobrecargando los métodos correspondientes.

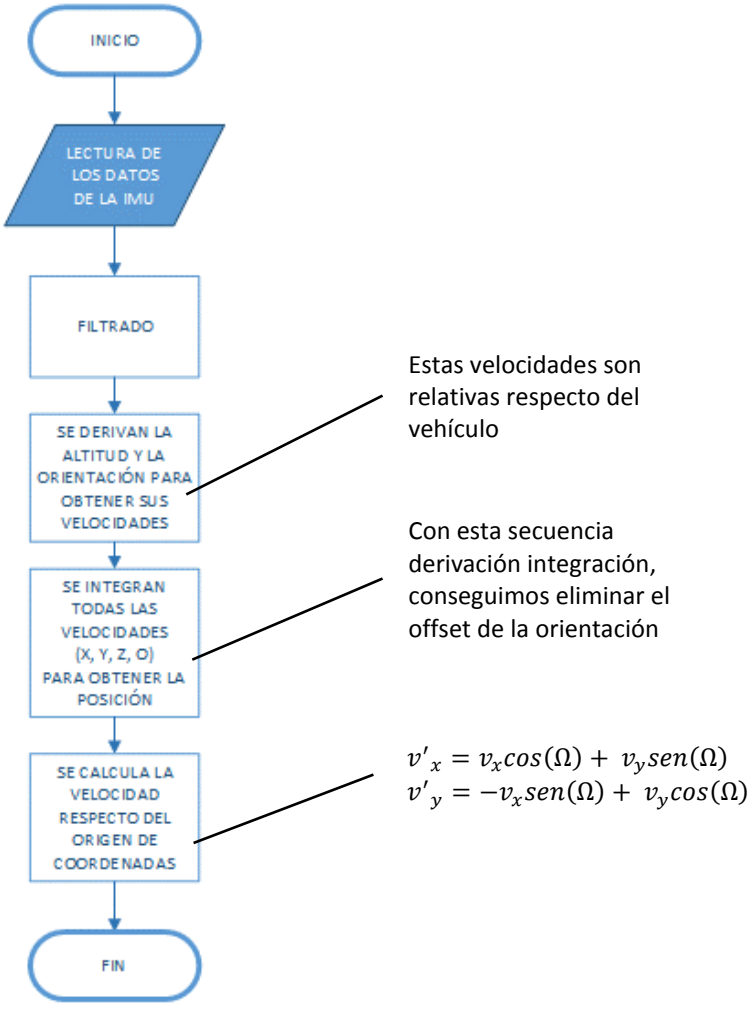


Ilustración 15.- Cálculo de estados mediante la IMU

Clase clsSystemInfo_IMAGE

Al igual que en la clase expuesta en el apartado precedente, esta clase hereda de clsSystemInfo, por lo que solo es necesario sobrecargar los métodos correspondientes.

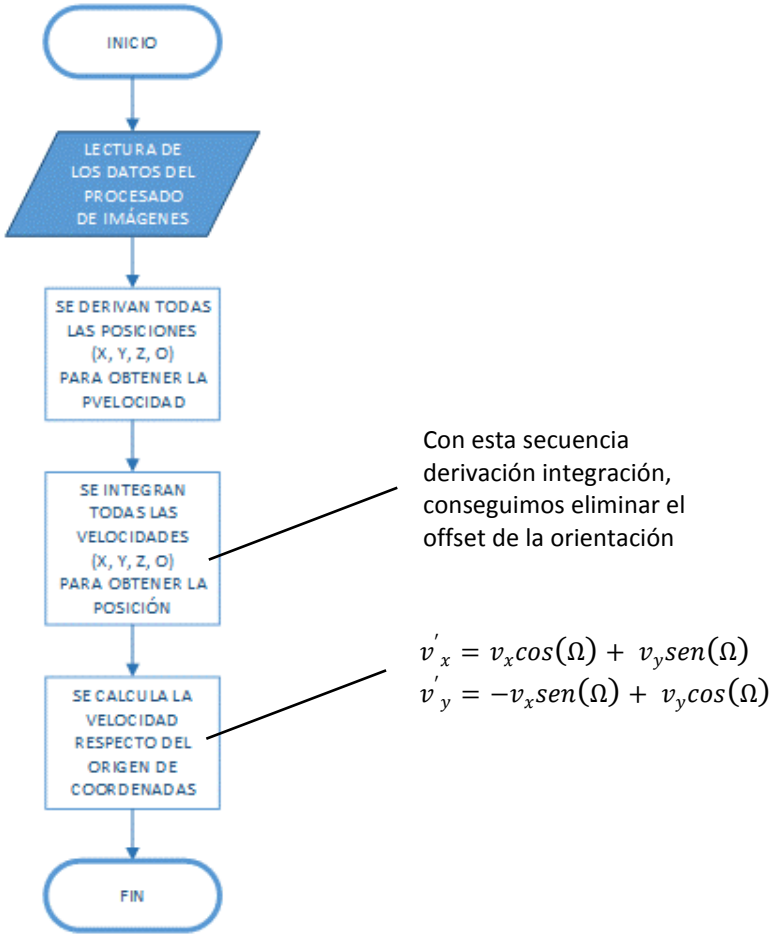


Ilustración 16.- Cálculo de estados mediante el procesamiento de imágenes

Clase clsControlSystem

Aquí se muestra la funcionalidad del método llamado por el evento de la clase clsSystemInfo, y que ordena la secuencia de control según el algoritmo descrito en el "Capítulo 3: Planteamiento general", "Algoritmo de Control Estático"

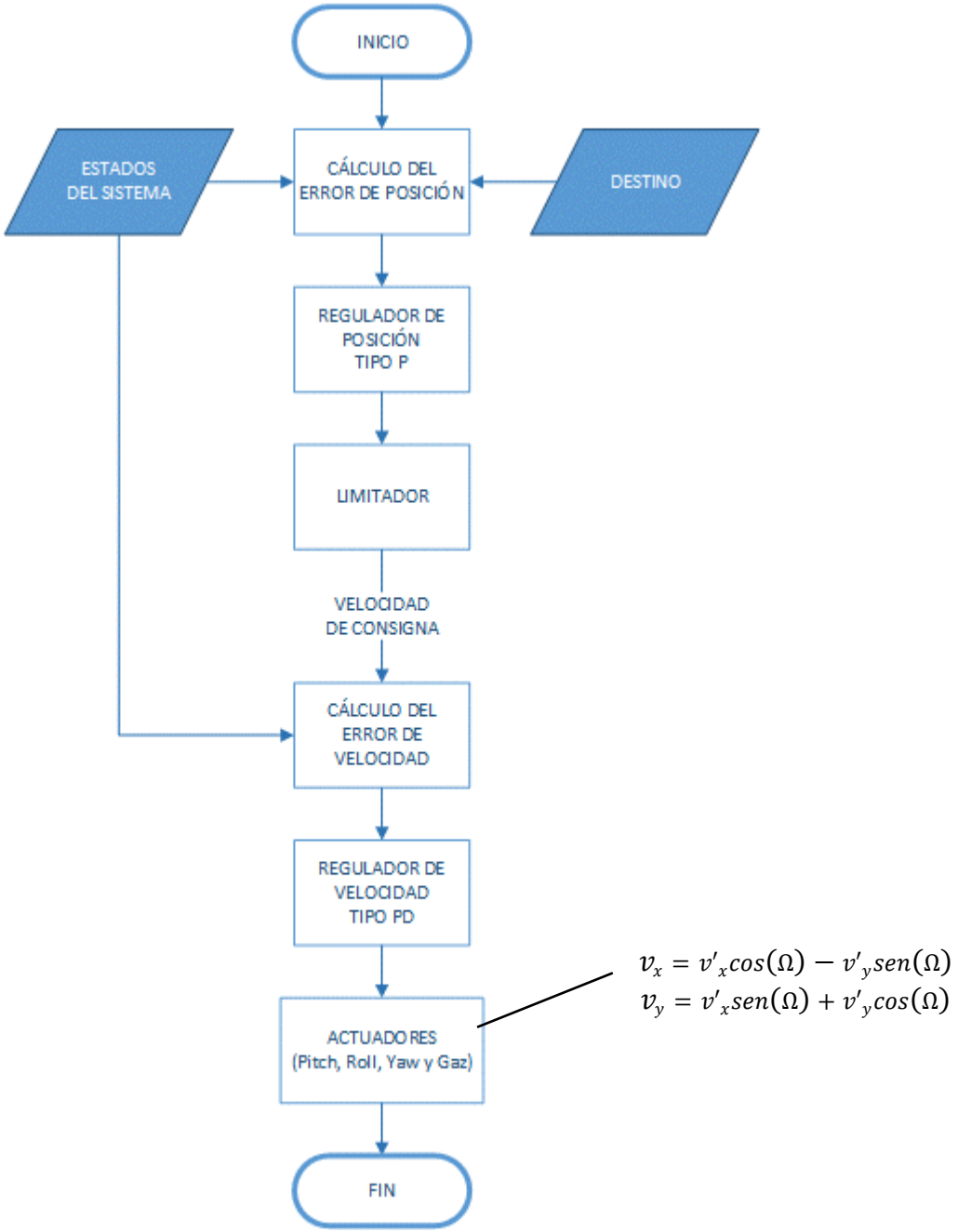


Ilustración 17.- Diagrama funcional clsControlSystem

Clase clsNavigationSystem

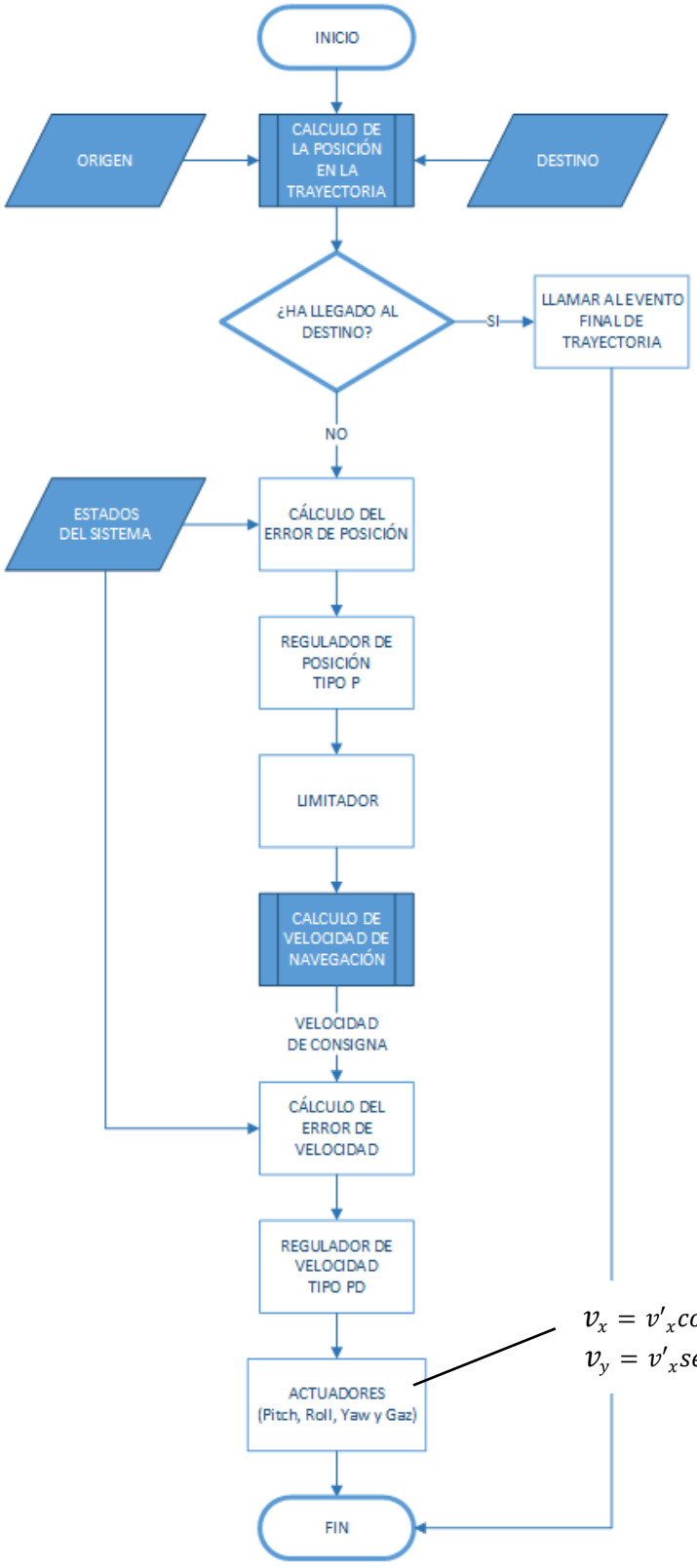


Ilustración 18.- Diagrama funcional clsNavigationSystem

Clase derivada de clsControlSystem, hereda sus miembros, y también su funcionalidad básica. En contraste con clsControlSystem, esta clase implementa el "Algoritmo de Control Dinámico", por lo que el método que secuencia el control está sobre cargado. Cuando se sobrecarga un método, el método de la clase base en realidad este no desaparece, sino queda oculto ya que el nuevo método de la clase derivada tiene mayor prioridad. Sin embargo es posible acceder al método de la clase base de forma explícita haciendo uso de la palabra clave *base*. De esta manera la clase clsNavigationSystem conserva la doble funcionalidad, haciendo uso del algoritmo estático o dinámico según convenga al usuario.

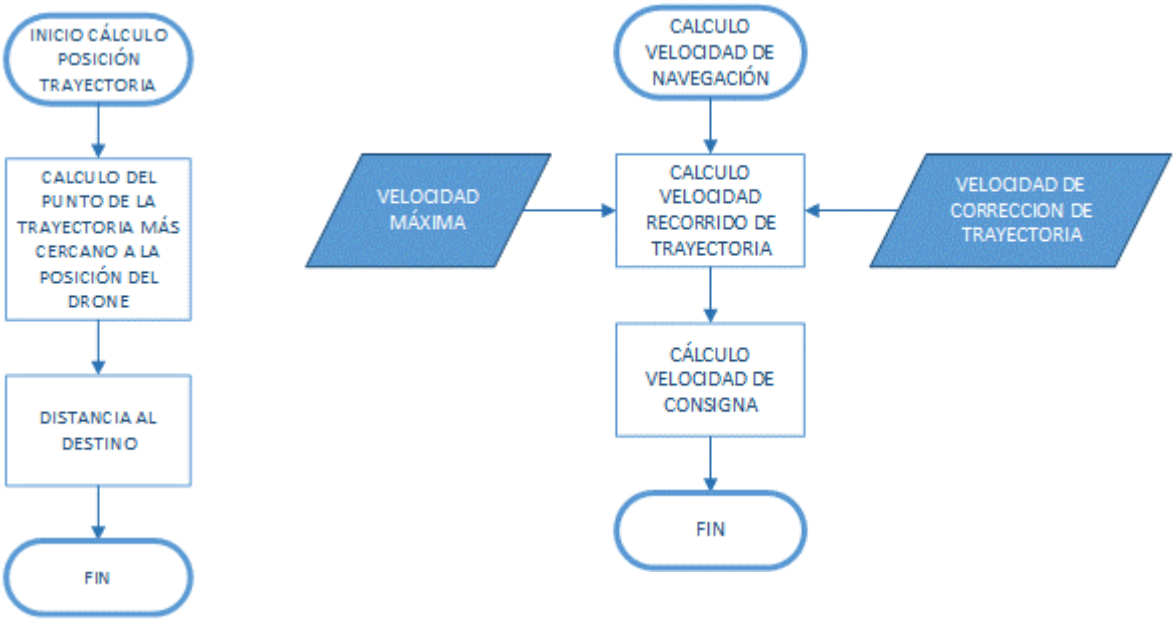


Ilustración 19.- Diagramas de cálculos de navegación

El desarrollo detallado de los cálculos para obtener el punto más cercano de la trayectoria a la que se encuentra el dron, se expone en el "Anexo B: Cálculos de navegación", en el apartado de "Cálculo de la trayectoria".

Del mismo modo, se pueden encontrar los cálculos conducentes a la obtención de la velocidad de recorrido de la trayectoria, explicados en el "Anexo B: Cálculos de navegación", en el apartado de "Cálculo de la velocidad".

Clase clsFlightPlan

Esta clase sirve para gestionar las trayectorias a realizar. Sus métodos presentan funcionalidades simples y concretas:

- FlyAway()

  Inicia el control de la trayectoria, y carga en el algoritmo el primer destino.

- NextPoint()

  Si todavía quedan puntos en la lista de destinos, carga el siguiente destino en el algoritmo, y continua realizando el control.

- AddPoint(double X, double Y, double Z, double O)

  Añade otro punto a la lista de trayectorias.

- DeletePoint()

  Situación opuesta al anterior componente. Este método borra la entrada de la lista seleccionada.

- ClearPoints()

  Borra todos los puntos.

- SavePoints(string f file)

  Guarda la lista actual de puntos en un archivo de texto plano

- LoadPoints(string file)

  Carga una lista de un archivo previamente salvado.

- connectTo(CheckedListBox)

  Vincular un objeto del formulario tipo *CheckedListBox* con el comando actual

- RefreshScreen()

  Actualiza la lista del objeto de formulario.

Clase clsAutomaticPilot

Esta clase deriva de clsCommands para heredar sus miembros. Esto es importante debido a que los miembros de clsCommands son usados como apoyo a la gestión de la navegación en las tareas de despegue, aterrizaje o suspensión en el aire.

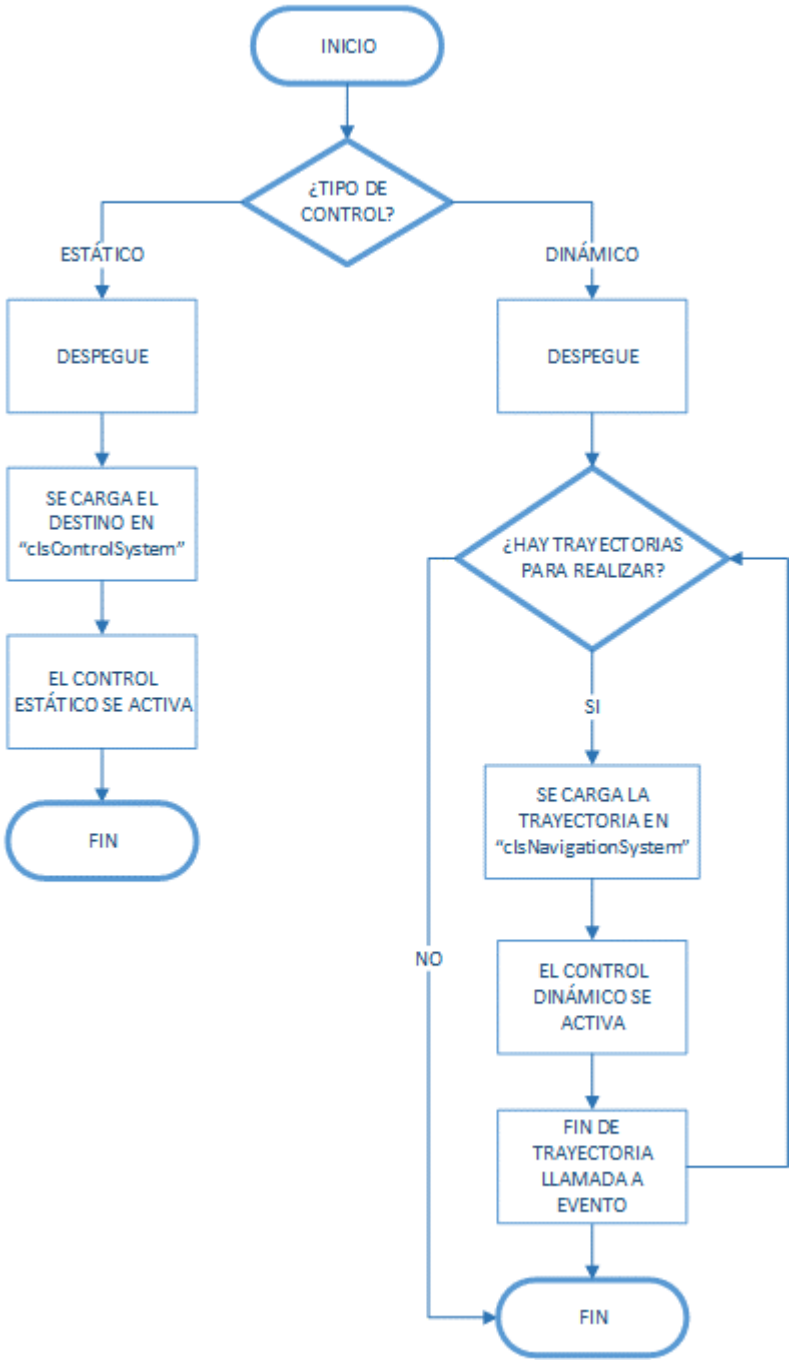


Ilustración 20.- Diagrama funcional clsAutomaticPilot

Clase clsP_I_D_Regulator

Implementación de un regulador PID. Para realizar los cálculos integral y derivativo, se recurre al uso de sendas clases clsIntegrator y clsDifferentiator. Los algoritmos de control no consideran la acción diferencial del regulador para que la respuesta no sea demasiado rápida y favorecer así el control. Sin embargo esta clase si considera esa posibilidad. El motivo que al escribir el código C# de este trabajo, se ha intentado que fuese lo más modular posible para permitir mejoras o añadidos futuros, o que pueda ser portado a otros proyectos totalmente distintos. Por este motivo, la clase clsP_I_D_Regulator representa un regulador PID genérico.

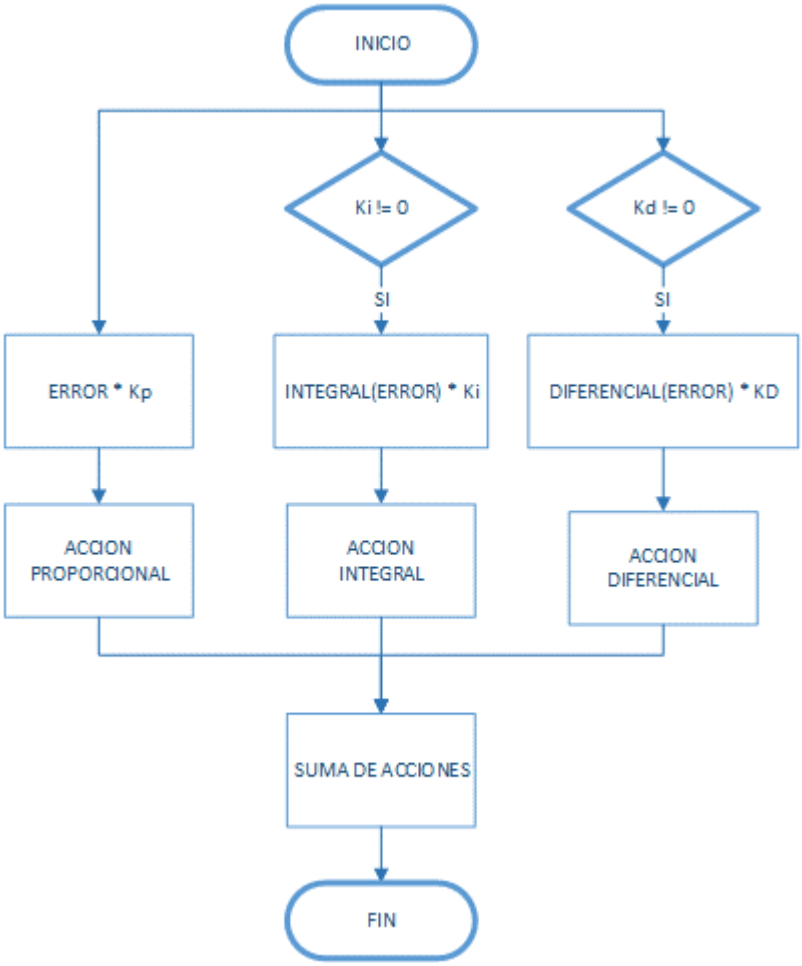


Ilustración 21.- Diagrama funcional clsP_I_D_Regulator

Clase clsDifferentiator

Clase con la funcionalidad de operar valores en diferencias.

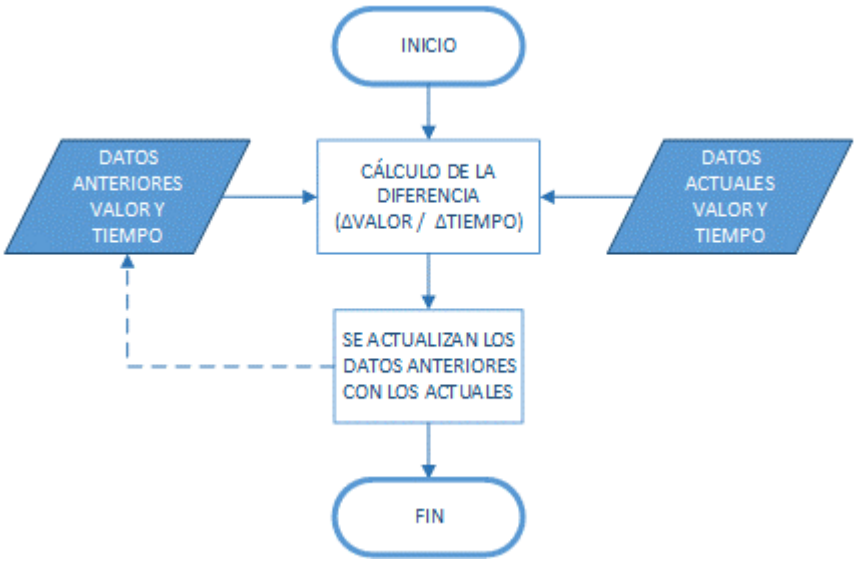


Ilustración 22.- Diagrama funcional clsDifferentiator

Nota.- La clase clsDiffrentiatorAngle es herencia de esta clase, por lo que comparte exactamente la misma funcionalidad. La única diferencia radica en que clsDifferentiatorAngle considera el caso en el cual el dron realiza un giro de 360º y se produce la incoherencia de que el ángulo salte de 360º a 0º de forma instantánea.

Clase clsIntegrator

Clase con la funcionalidad de integrar los valores dados.

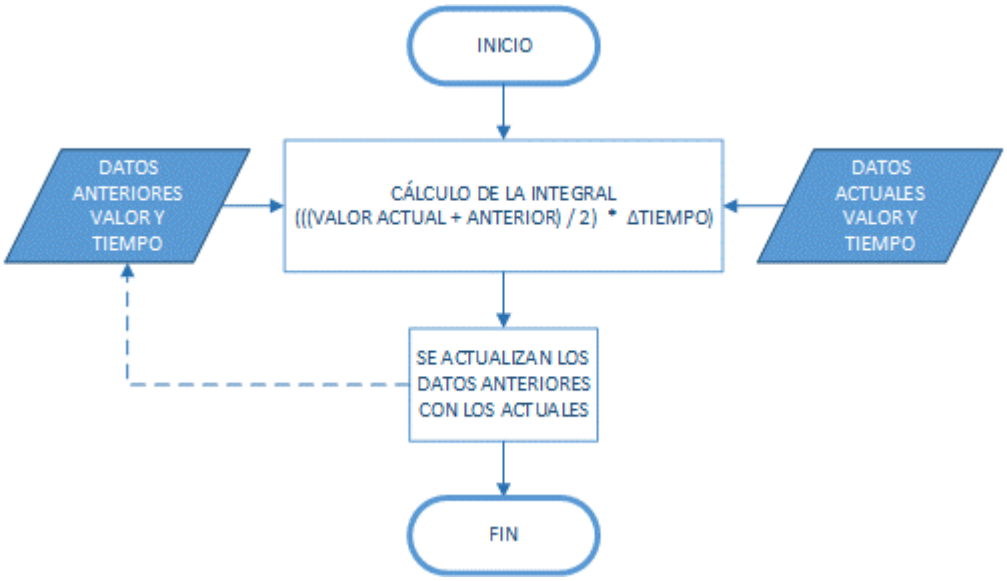


Ilustración 23.- Diagrama funcional clsIntegrator

Nota.- La clase clsIntegratorAngle es herencia de esta clase, por lo que comparte exactamente la misma funcionalidad. La única diferencia radica en que clsIntegratorAngle trata el valor de salida para que solo se mueva entre los valores de (-π, π]

Clase clsFilter

La funcionalidad de esta clase es un filtro promediador, el cual se comporta como un filtro paso bajo, reduciendo el ruido de las medidas, he introduciendo cierto retraso en la señal.

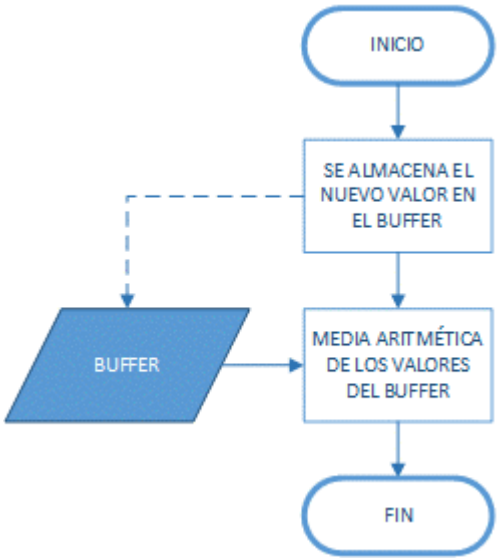


Ilustración 24.- Diagrama funcional de la clase clsFilter

FUNCIONALIDADES DEL FORMULARIO

Grupo de controles "Status"

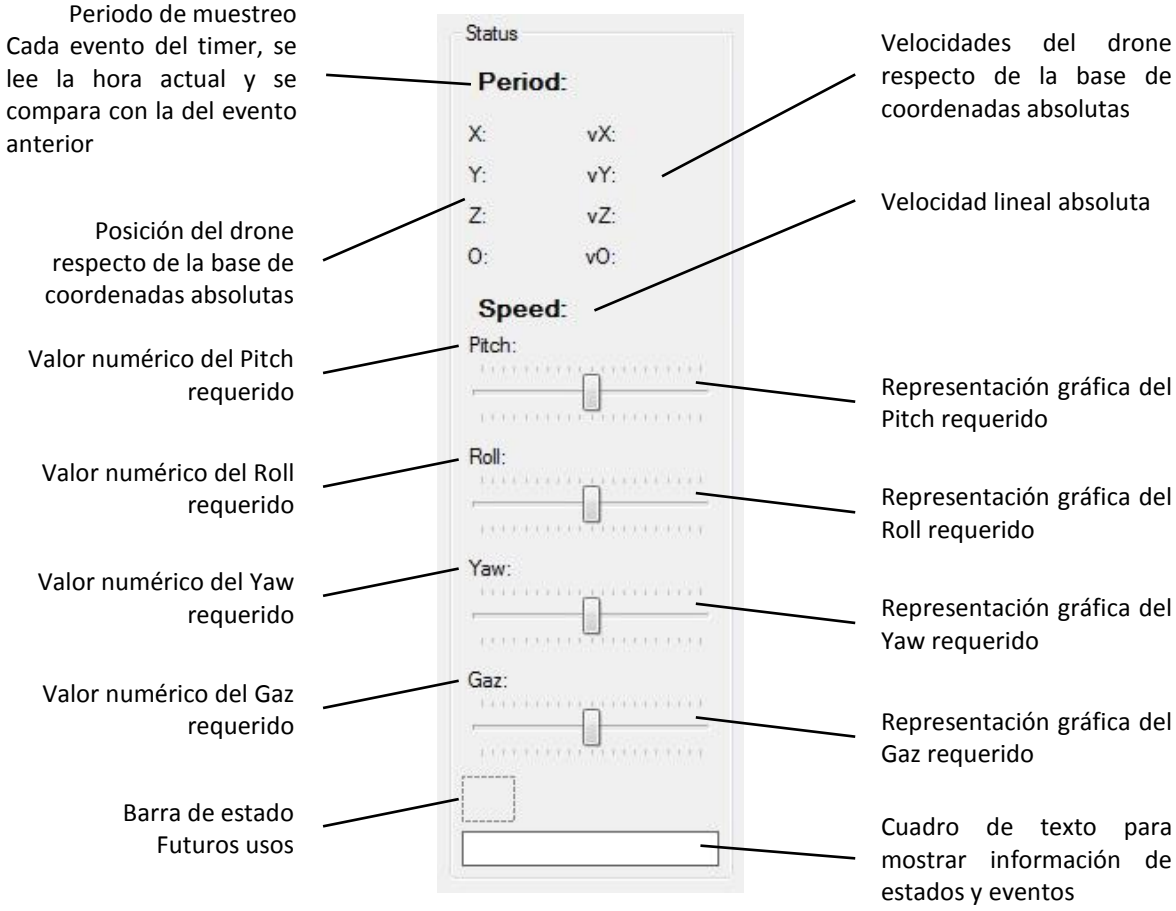


Ilustración 25.- Controles del grupo "Status" del formulario

Cuando los valores de Pitch, Roll, Yaw o Gaz superan el 90% del valor máximo, los controles *slider* que dan muestra gráfica de su situación, se iluminan en rojo para informar al usuario de la situación límite.

El cuadro de texto proporciona información del sistema, que ayuda a validar el correcto funcionamiento, como información de la trayectoria o del estado del drone.

Grupo de controles "Trayectory"

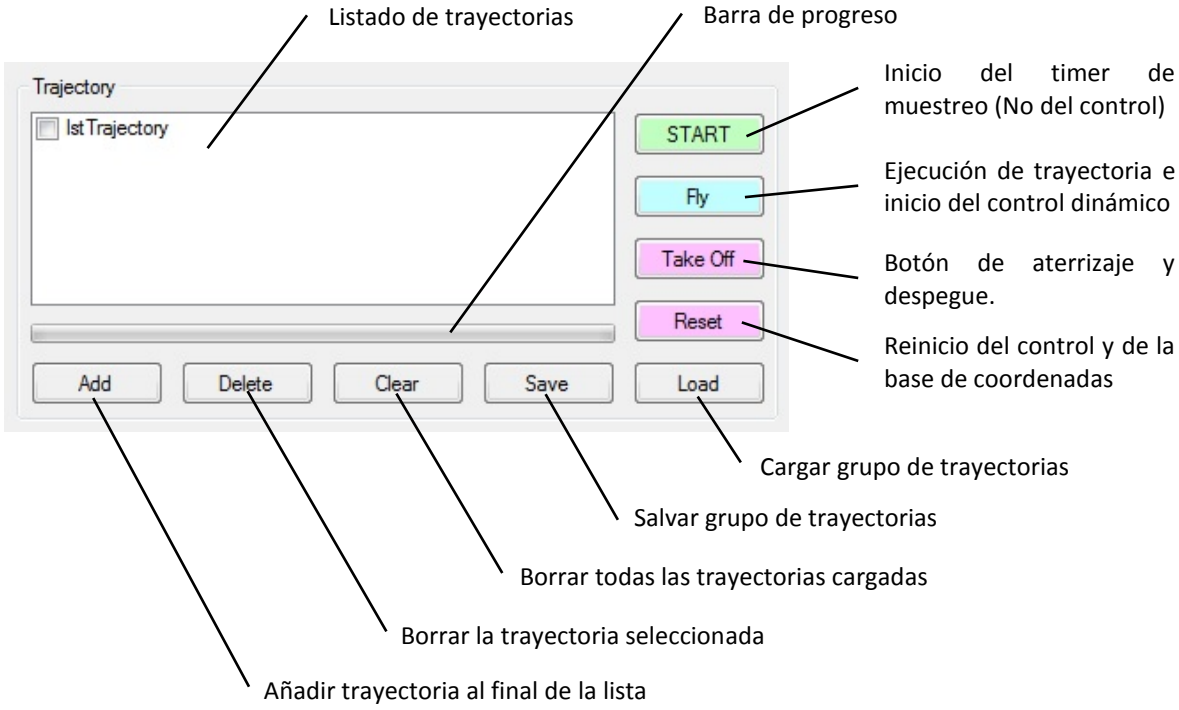


Ilustración 26.- Controles del grupo "Trajectory" del formulario

El control "lstTrayectory" que muestra por pantalla el listado de trayectorias, ha de vincularse a la clase "clsFlightPlan", ya que son las funcionalidades de esta clase las que gestionan este control. Tanto es así, que los controles gráficos "Add", "Delete", "Clear", "Save" y "Load", lo único que hacen es llamar a métodos de la clase "clsFlightPlan". Esto nos sirve para evitar errores que se pueden producir cuando los controles están desligados de los atributos de las clases y hay que realizar la doble acción de modificar el control y el atributo. Por ejemplo, si al pulsar el botón "Add", se añadiese un nuevo texto al control "lstTrayectory", pero no se añadiese la trayectoria a la clase clsFlightPlan, se produciría un comportamiento inesperado del drone.

La barra de progreso nos muestra el progreso del drone en la realización de cada trayectoria.

El botón "Take Off" modifica su etiqueta alternando entre "Take Off" y "Landing", según la situación correspondiente.

Al pulsar sobre el botón "Reset", se reinician los algoritmos de control y la base de coordenadas a la posición y orientación actual de drone.

Grupo de controles "Coordinates"

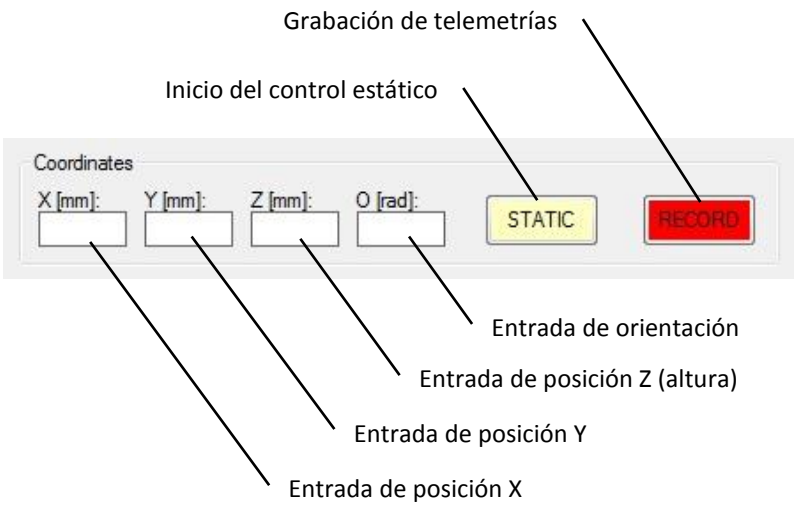


Ilustración 27.- Controles del grupo "Coordinates" del formulario

La entrada de coordenadas sirve tanto para ejecutar el algoritmo estático, como para introducir nuevas trayectorias en la lista y ejecutarlas posteriormente con el algoritmo dinámico.

En las entradas de texto, se controla la escritura, de tal manera que no se pueden introducir caracteres distintos de los numéricos, evitándose así posibles errores.

Pulsando el botón "RECORD" se pueden guardar las telemetrías. Una ventana de guardado aparece, solicitándonos nombre para el archivo, y ubicación. Una vez pulsado, el tono del botón cambia a rojo como en la ilustración, para indicarnos que la grabación de datos esta activa. Para desactivarla solo hay que pulsar de nuevo el mismo botón.

Grupo de controles "Parameters"

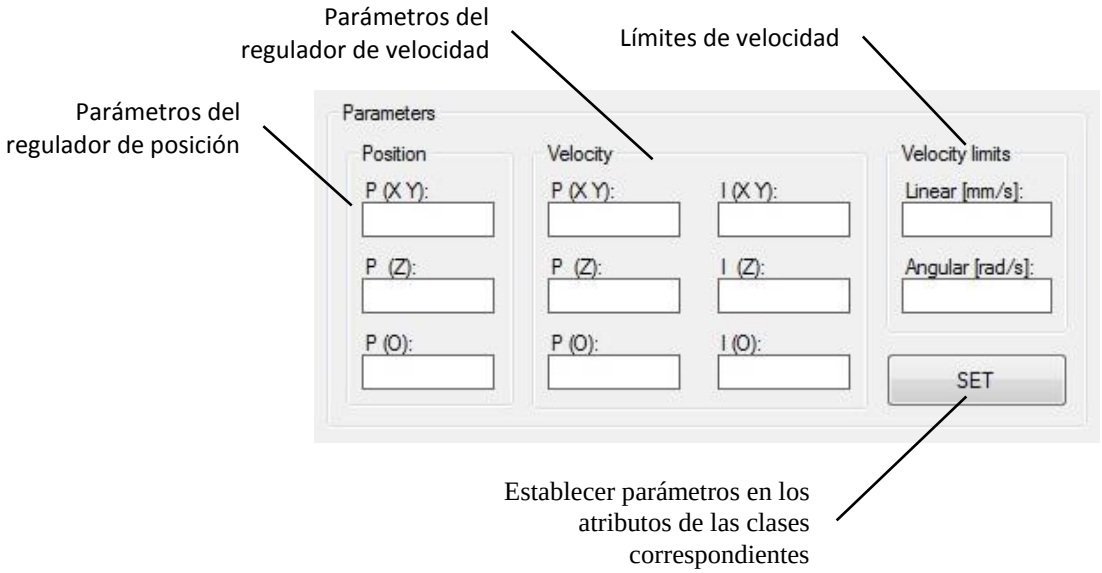


Ilustración 28.- Controles del grupo "Parameters" del formulario

En las entradas de texto, se controla la escritura, de tal manera que no se pueden introducir caracteres distintos de los numéricos, evitándose así posibles errores.

Al pulsar el botón "SET", los parámetros de cada cuadro de texto se introducen en el atributo de su clase correspondiente. Acto seguido el programa verifica la validez de los valores introducidos, comparando el valor presente con el atributo de la clase, con el valor del cuadro de texto. Si estos valores coinciden, significa que el parámetro ha sido introducido convenientemente y sin fallos, cambiado el color de fondo del cuadro de texto validado a verde, para representar su conveniencia. Si se produjese un fallo en la introducción de algún valor, el cuadro de texto correspondiente se colorearía de rojo indicativo.

Al modificar el texto de algún cuadro, su fondo vuelve al tono blanco, representando que todavía no ha sido validado.

Introducir los parámetros en los cuadros de texto cada vez que se abre el formulario, es una tarea tediosa. Por ello, al pulsar el botón "SET", automáticamente se crea un archivo de configuración con los valores de los parámetros que será guardado en la misma ubicación que el programa en ejecución.

De esta forma, al abrir de nuevo el formulario, el programa accede al archivo de configuración, guarda cada parámetro en su cuadro de texto, y finalmente los valida.

LLAMADAS A CONTROLES EN HILO SEGURO

Visual C# funciona en un entorno manejado, lo que significa que vigila y asiste al usuario en ciertas acciones. Escribir un código de un modo que no sea seguro para el compilador de C# genera excepciones que han de ser tratadas.

En concreto, C# origina una excepción cuando un subproceso de un hilo de ejecución intenta manipular un control que ha sido instanciado en un hilo de ejecución distinto. Esta problemática apareció en el desarrollo del código debido a que para actualizar el entorno gráfico, se hace uso de un timer que lo refresca periódicamente. Al instanciar el timer, se crea un hilo particular de ejecución, que incluyen las funciones a las que llama el timer. Este comportanmiento del timer mejora el rendimiento, ya que se vale de las bondades de la ejecución multihilo, sin embargo, acceder a un control desde otro hilo de ejecución genera inestabilidades en la ejecución, que C# con tolera, y han de ser tratadas.

Este es un problema típico y tiene varias maneras de solucionarse. Para el desarrollo de este programa se hace uso de la función *Invoke*:

```
private delegate void LlamadaControl(tipoparametro parametro);

private void FuncionLlamada(tipoparametro parametro)
{
	if (control.InvokeRequired)
	{
		LlamadaControl d = new LlamadaControl(FuncionLlamada);
		Invoke(d, new object[] { parametro });
	}
	else
	{
		control.Atributo = parametro;
	}
}
```

De esta manera se pasa el parámetro al hilo de ejecución del control, y se modifica el atributo del control en su correspondiente hilo. [5]

CAPÍTULO 5:
PRUEBAS Y RESULTADOS

- ***Introducción***
- ***Resultados***
- ***Análisis***

INTRODUCCIÓN

Para poder analizar los resultados, es imprescindible un método de volcado de datos, que nos permita un tratamiento posterior. Con este objeto ya ha añadido al programa la funcionalidad de crear telemetrías. La salida de la telemetría en un archivo de texto plano, con una cabecera con datos básico de guardado y parámetros de los reguladores, y los datos en bruto, que representan cincuenta variables muestreadas en cada ciclo de reloj.

De las diversas pruebas que se han realizado, se han obtenido más de 20 telemetrías, la mayoría de la cuales se han descartado al encontrarse bugs en la programación que las invalidaban. Estos bugs se han ido depurando hasta conseguir una programación estable que permita obtener telemetrías representativas aptas para sacar conclusiones.

Los resultados que expone este capítulo pertenecen a la telemetría número 19, realizada el día 23 de Septiembre de 2016 por Jorge Sánchez García, autor de este trabajo, con la colaboración de Pablo Alías Mateos, alumno también de la Universidad Carlos III de Madrid. Las condiciones climáticas eran favorables, con cielo despejado y leve brisa a intervalos racheada. La ubicación se encuentra en uno de los parkings del Estadio Municipal Butarque, Leganés.

La prueba consiste en realizar una figura con forma de aspa. Las coordenadas de la figura se asientan en un plano a un metro del suelo. En pruebas precedentes se ha podido comprobar que al realizar trayectorias en altura, las rachas de viento dificultan el control sobremanera.

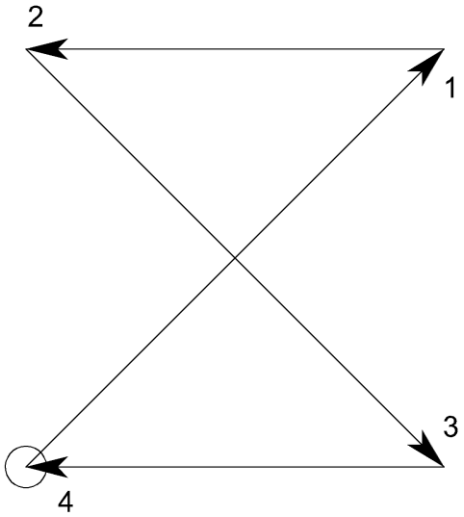


Ilustración 29.- Recorrido trayectoria de prueba

1) X=3000 Y=3000 Z=1000 O=0
2) X=0 Y=3000 Z=1000 O=0
3) X=3000 Y=0 Z=1000 O=0
4) X=0 Y=0 Z=1000 O=0

Las trayectorias se realizan con el algoritmo de control dinámico, y al concluir, el drone se mantiene unos segundos en el punto 4 por medio del algoritmo estático, con el objeto de hacer mediciones de ambos algoritmos.

Al concluir el punto 4, se le ordena al drone que gire sobre su eje π radianes, en control estático, como se acaba de mencionar.

Los valores de los parámetros de control son los siguientes:

- Regulador de la posición. Contantes proporcionales.

  XY = 0'5 Z = 0,3 O = 0,5

- Regulador de la velocidad. Contantes proporcionales.

  XY = 0'0005 Z = 0,001 O = 0,1

- Regulador de la velocidad. Contantes integrales.

  XY = 0'0001 Z = 0,001 O = 0,1

Los valores de los límites de velocidad son los siguientes:

- Límite de velocidad lineal

  300 mm/s

- Límite de velocidad angular

  2 rad/s

RESULTADOS

Conclusión general de la trayectoria

En este gráfico se muestra en planta la trayectoria realizada superpuesta con la trayectoria ideal. Como se ha comentado anteriormente, no hay desplazamientos en altura, aunque la altura también está sujeta a control.

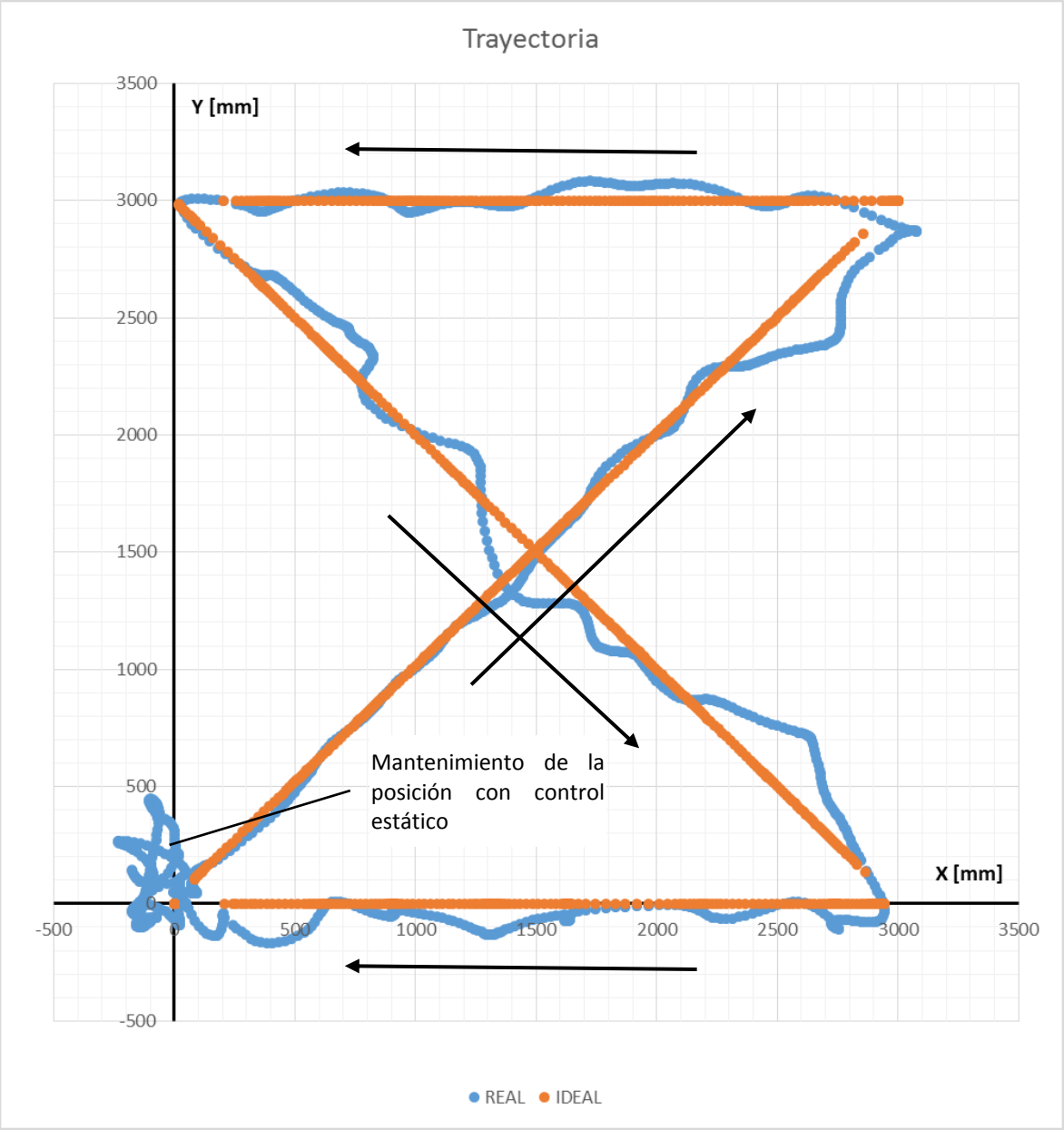


Ilustración 30.- Trayectoria muestreada

Velocidades absolutas

El primer gráfico muestra el módulo de velocidad lineal absoluta en cada momento, junto con una recta de regresión.

El segundo gráfico muestra la velocidad de giro. Como la trayectoria se ha realizado sin giro, la velocidad oscila en torno al valor cero. Las oscilaciones se deben a la acción de control que intente contrarrestar las perturbaciones.

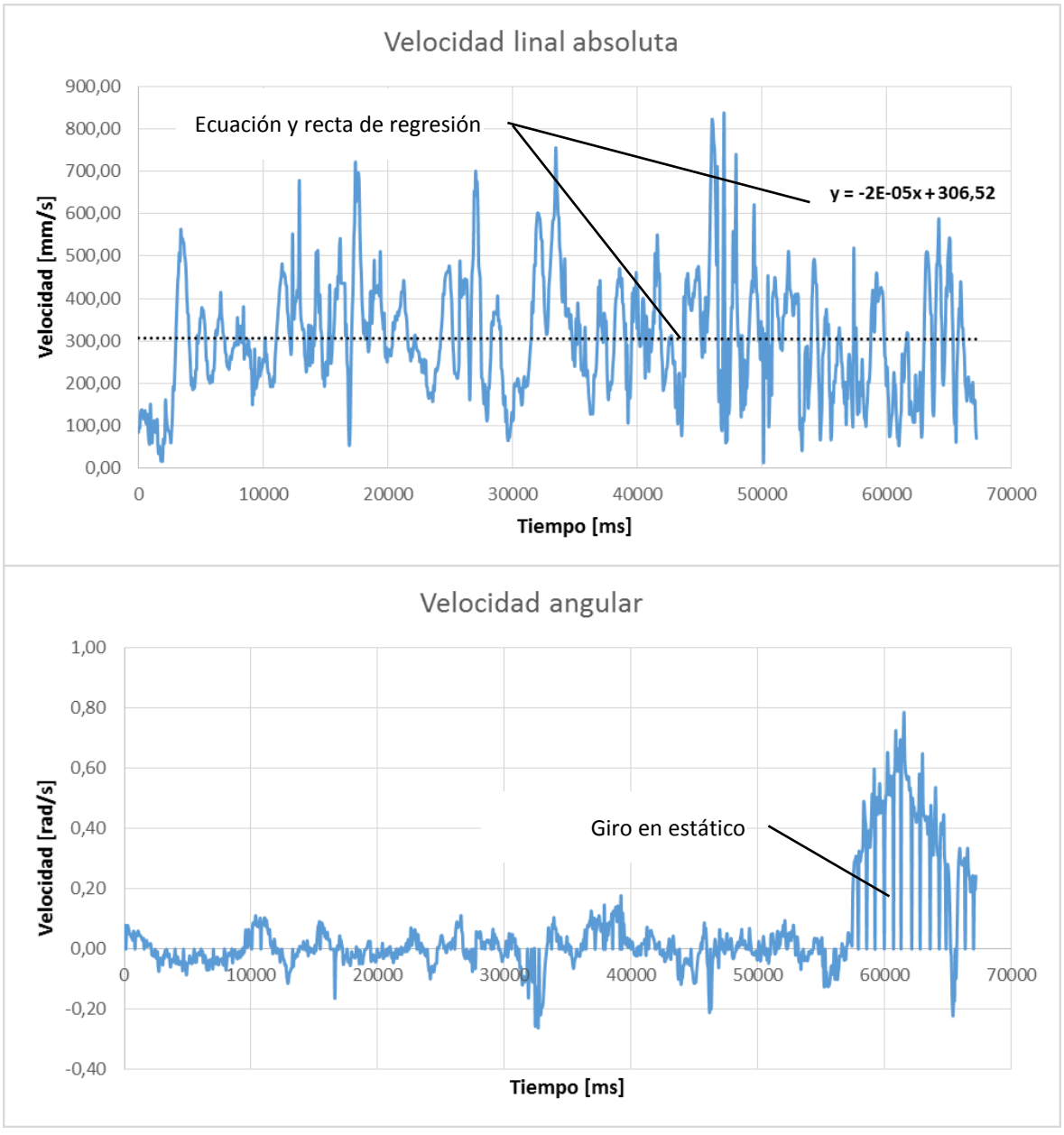


Ilustración 31.- Velocidades absolutas lineal y angular

Control de la posición

En los siguientes gráficos se puede apreciar la comparación entre la posición real del drone y su posición objetivo o de consigna, particularizando para cada eje.

Superpuesto a estas gráficas, se encuentra el error de posición para cada eje. La escala de la derecha pertenece al error de posición.

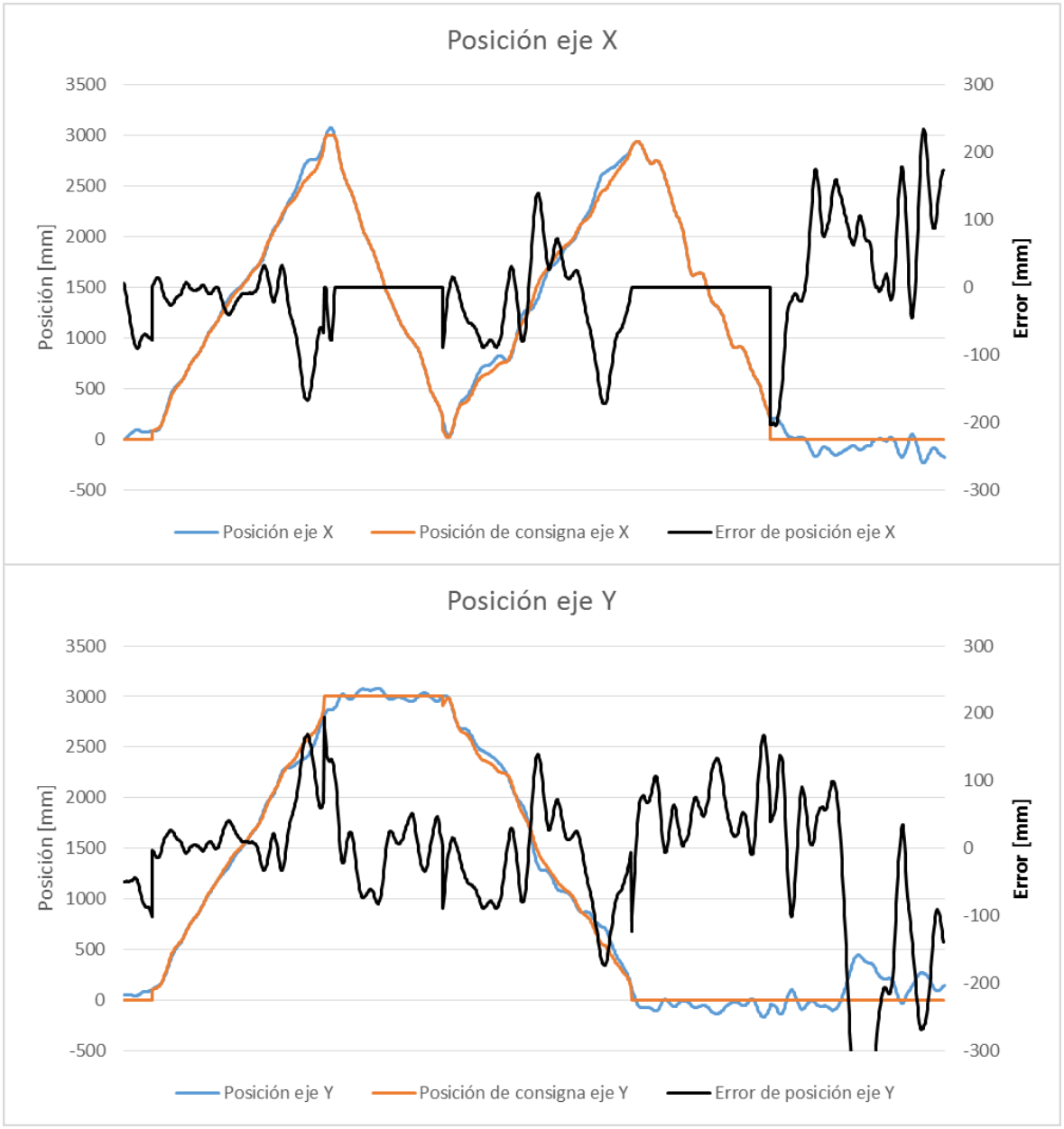


Ilustración 32.- Posición muestreada ejes X e Y

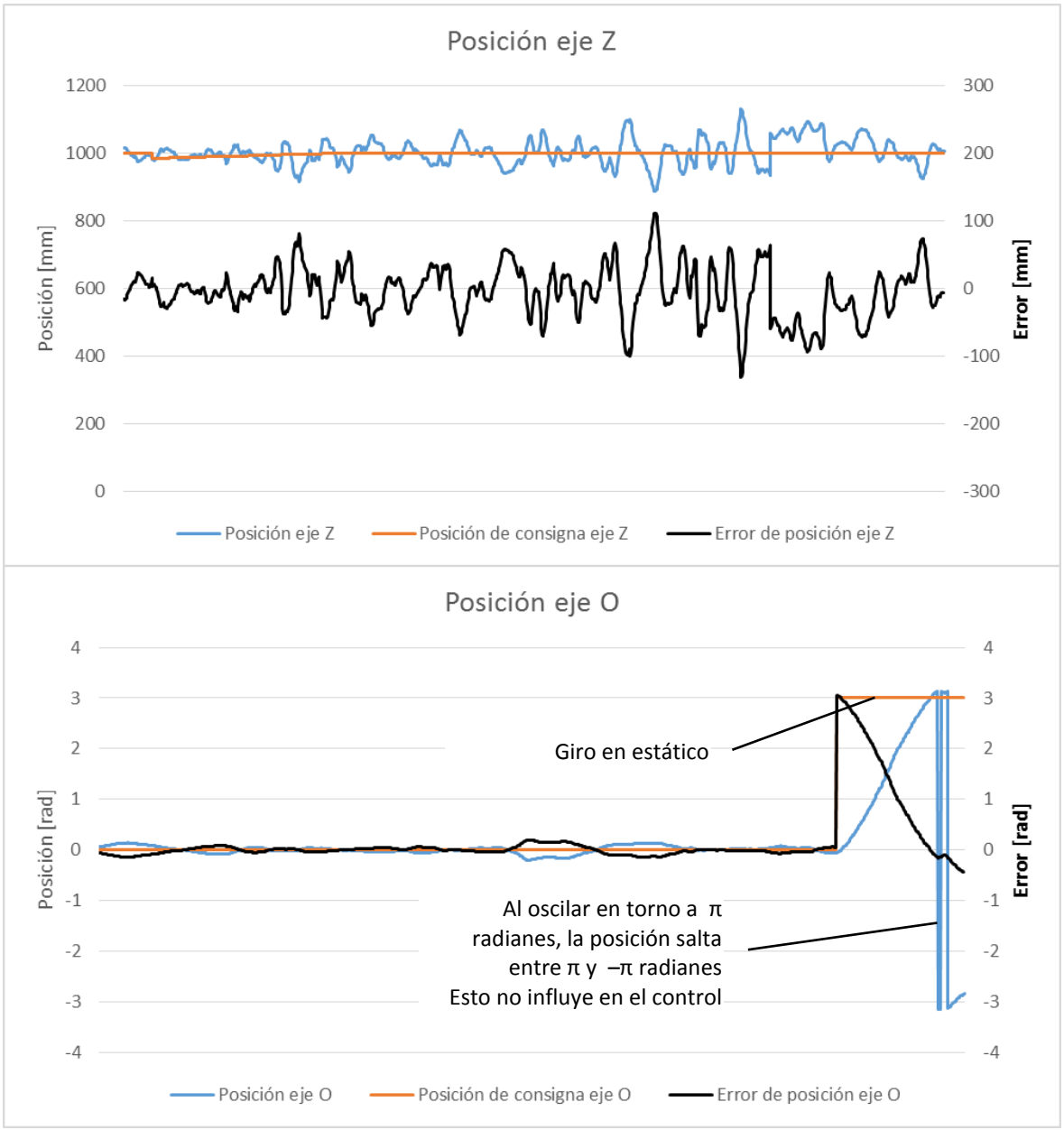


Ilustración 33.- Posición muestreada ejes Z y O

Control de la velocidad

En los siguientes gráficos se puede apreciar la comparación entre la velocidad real del drone y su velocidad objetivo o de consigna, particularizando para cada eje.

Superpuesto a estas gráficas, se encuentra el error de velocidad para cada eje. La escala de la derecha pertenece al error de velocidad.

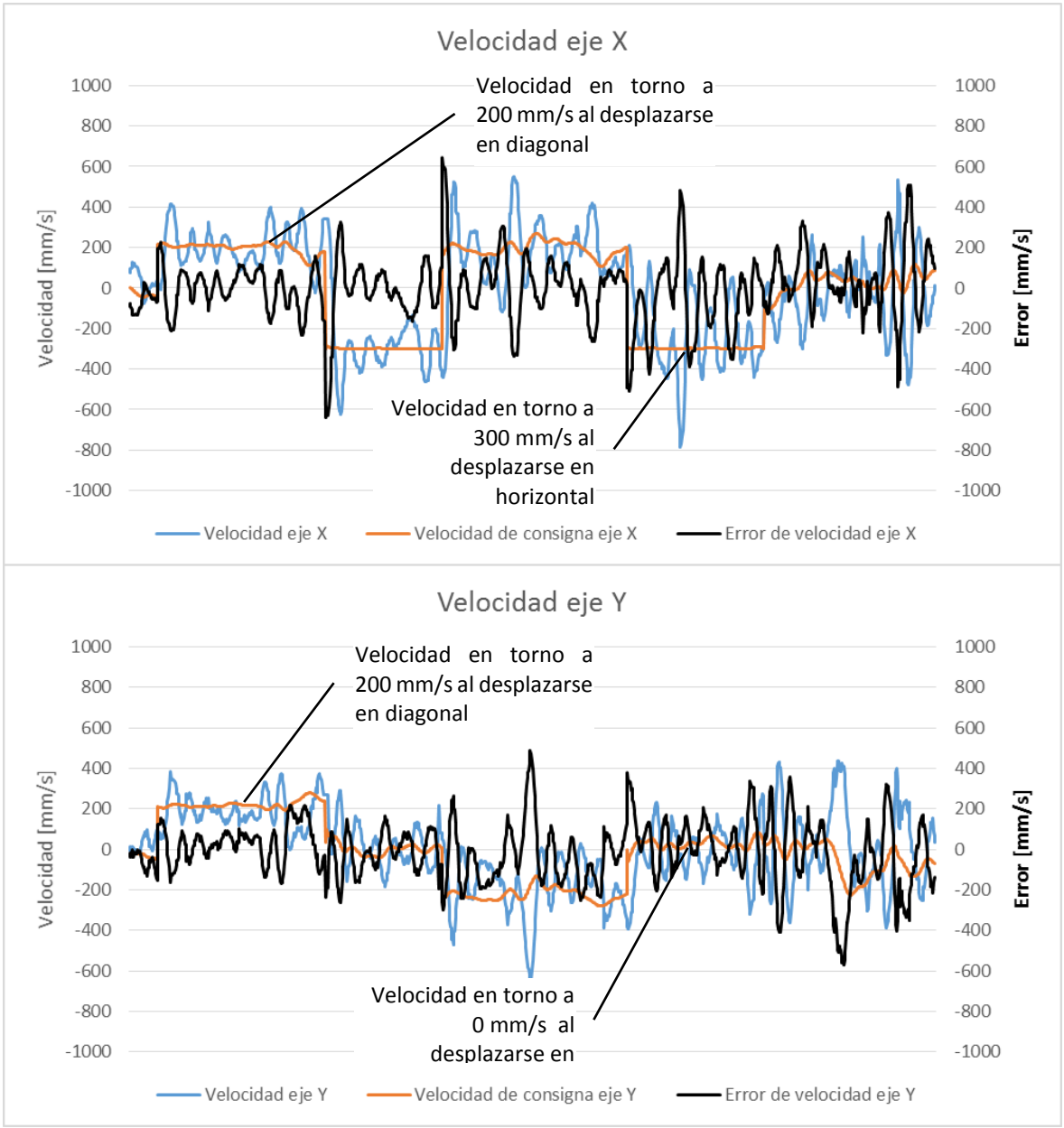


Ilustración 34.- Velocidad muestreada ejes X e Y

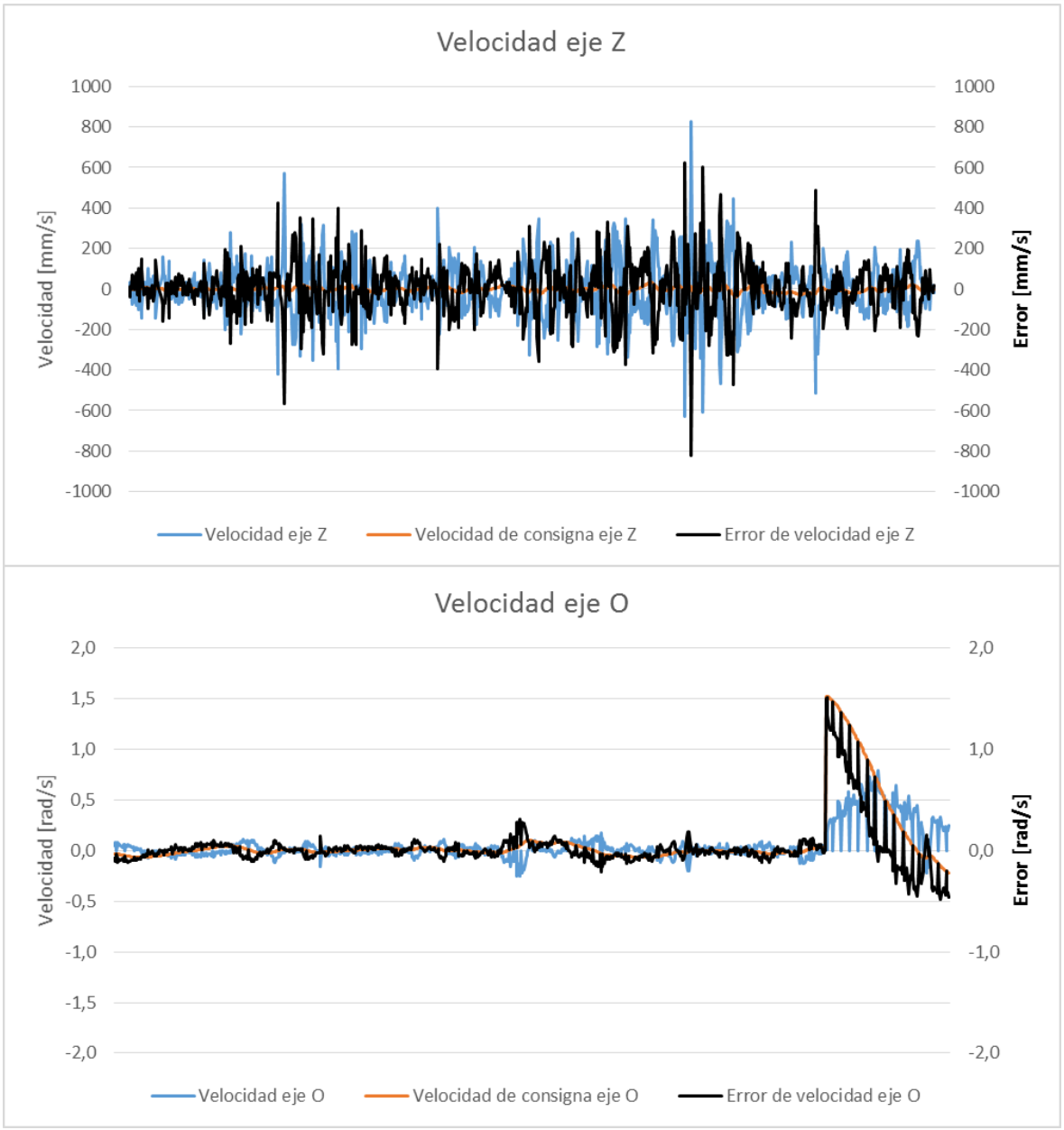


Ilustración 35.- Velocidad muestreada ejes Z y O

Acción de los actuadores

Gráficas de las órdenes de control que reciben los actuadores individualizadas para cada eje.

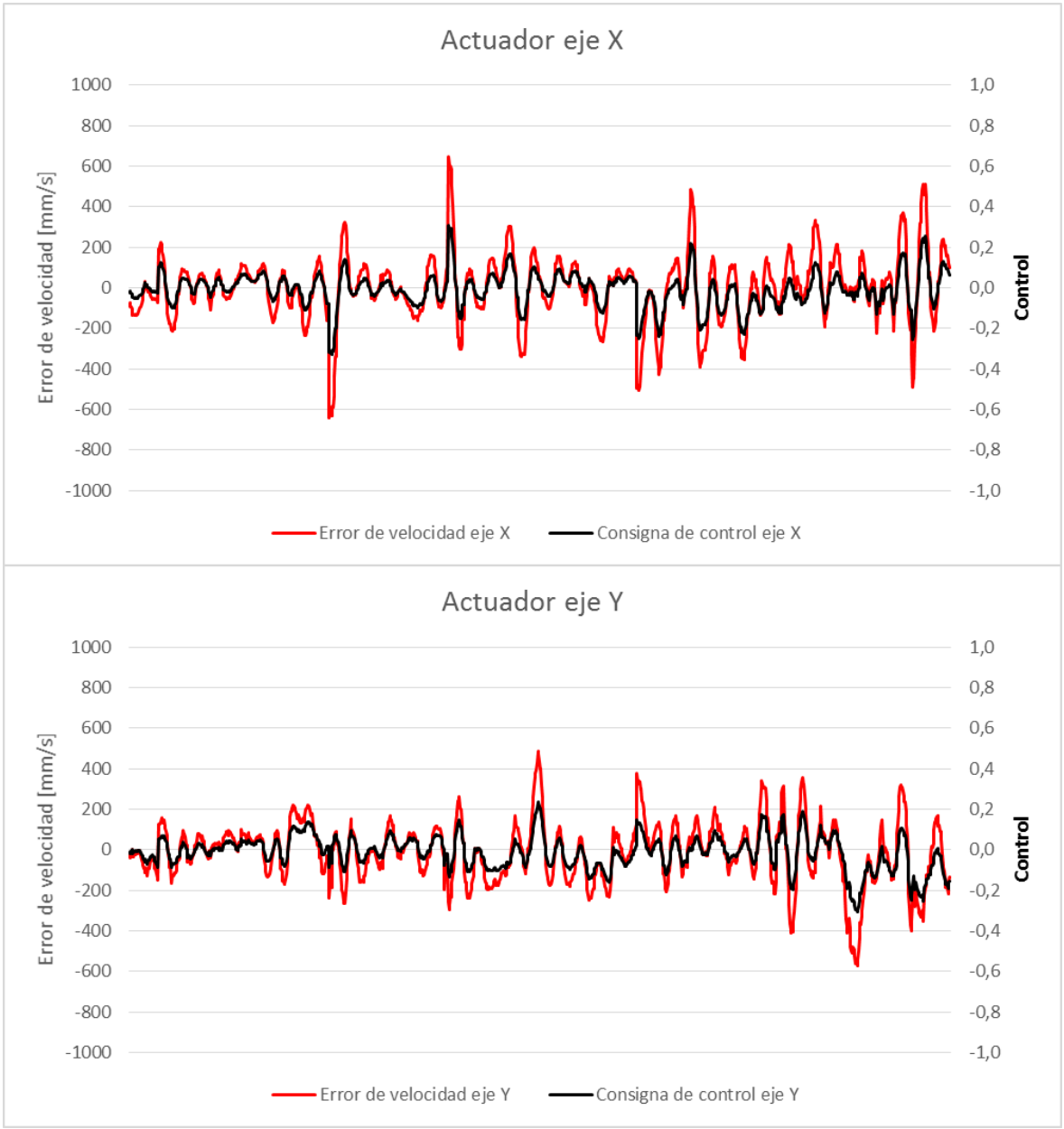


Ilustración 36.- Acción de los actuadores ejes X e Y

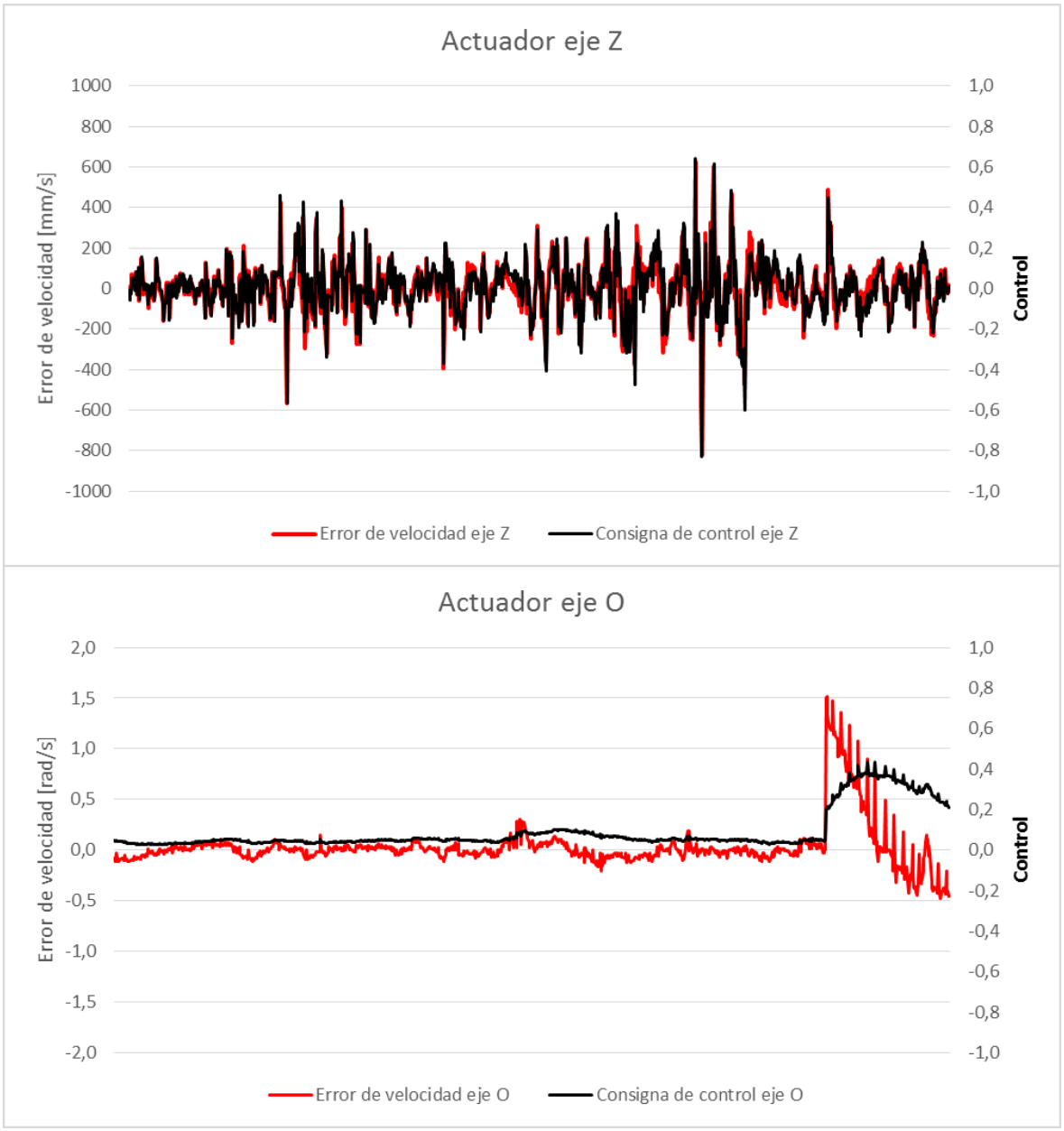


Ilustración 37.- Acción de los actuadores ejes Z y O

Cabe mencionar que las consignas de control no representan directamente ordenas a los actuadores, ya que para traducir las consignas en los ejes X e Y en valores de Pitch y Roll, hay que tener en cuenta la orientación. Además, deben ser limitadas al rango de (-1, 1] para ser aplicables como ordenes de Pitch, Roll, Yaw y Gaz. Como salvedad, las consignas de control de los ejes Z y O si representan en este caso sendas ordenas de Gaz y Yaw, ya que sus valores no exceden del rango mencionado.

ANÁLISIS

Como se puede apreciar en la *Ilustración 30.- Trayectoria Muestreada*, el drone sigue con bastante fidelidad la trayectoria introducida, lo que en sí mismo es representativo de un correcto funcionamiento del algoritmo de control. En esta misma ilustración se hacen evidentes las oscilaciones del drone en torno a la trayectoria, lo cual es normal, puesto que la prueba se realizó al aire libre, y una ligera brisa es suficiente para perturbar la posición de un drone. Lo significativo es que el drone responda a esas perturbaciones volviendo a la trayectoria y continuando su camino.

En la *Ilustración 31.- Velocidades absolutas lineal y angular*, centrándonos primero en la gráfica de velocidad lineal, vemos como la velocidad presenta altibajos. Sin embargo, al trazar la recta de regresión, observamos como esta se sitúa muy próxima a los 300 mm/s, que es el límite de velocidad lineal establecido para esta prueba. Lo que significa que aunque el sistema oscile, estas oscilaciones son normales por las perturbaciones antes mencionadas. Lo significativo es que el sistema en ningún momento se descontrola, y siempre se mueve en torno a su valor de referencia.

En la misma ilustración, fijándonos en la gráfica de la velocidad angular, vemos como la velocidad se mantiene en torno a cero, lo cual es correcto, puesto que las trayectorias no tienen programado giro alguno. Sin embargo si vemos al final de la gráfica como la velocidad varía debido a que al concluir el recorrido, se le ordena al drone que gire media vuelta sobre su eje. Las líneas verticales que aparecen en esta gráfica son debidas a que el tiempo de muestreo es más pequeño de lo debido, y cada cierto tiempo, dos medidas consecutivas registran el mismo valor, lo que en términos de velocidad significa que la velocidad es cero, y así la gráfica cae hasta el origen a intervalos periódicos.

De las ilustraciones 32 y 33 sobre la posición del drone, podemos inferir lo mismo que de la ilustración 30 sobre la trayectoria, ya que representan el mismo análisis, pero particularizado para cada eje. Observamos la evolución del drone en cada eje respecto su posición ideal, y el error que este desfase origina. Nuevamente, el drone se mueve en torno a su posición de referencia, lo cual es adecuado.

En las velocidades mostradas en las ilustraciones 34 y 35, se observa como la velocidad va cambiando debido al trazado de las distintas trayectorias, y como el drone intenta seguir esta trayectoria

Un análisis igual de positivo puede hacerse de las gráficas de los actuadores

CAPÍTULO 6: CONCLUSIONES Y TRABAJOS FUTUROS

- ***Incidencias***
- ***Conclusiones***
- ***Trabajos futuros***

INCIDENCIAS

La primera incidencia con la que nos hemos topado, es con el tiempo de muestreo. El timer del control está configurado para ejecutarse cada 50m (este valor puede ser modificado en tiempo de compilación), sin embargo, al no ser Windows un sistema determinista, este periodo de muestre se eleva hasta los 60ms. La elección de este periodo de muestreo es puramente experimental, y es que durante las pruebas se ha observado que si se reduce el tiempo de muestre, se repiten mediciones por que no se le otorga tiempo a los datos a actualizarse. De hecho este fenómeno ya se produce a 50ms, como se ha mencionado en el apartado "Análisis" del capítulo 5º "Pruebas y resultados". La problemática radica en que 50ms de tiempo de muestro es de un orden similar al tiempo de respuesta del drone ante perturbaciones externas, lo que imposibilita un control más fino.

Otra incidencia es debida al carácter "low cost" de las mediciones de la IMU, cuyos valores son demasiado volátiles. Se ha intentado aplicarles a estos valores un filtro promediador para intentar reducir la incertidumbre, pero lo único que se ha conseguido es retrasar la señal, ya que al ser el tiempo de muestre tan grande, incluso usando un buffer pequeño para el filtro, el tiempo de refresco del filtro es superior al tiempo de respuesta del sistema; lo que origina incluso mayor problemática que si no se usa el filtro.

Por último a destacar, las dificultades para mantener una conexión fiable entre el drone y el ordenador. Ya que es frecuente la pérdida repentina de conexión, o que manteniéndose conectado, las medidas se congelen, lo que provoca un mal comportamiento en el drone.

CONCLUSIONES

A tenor de las incidencias anteriormente mencionadas, podemos concluir que el modelo ARDrone 2.0 no es adecuado para el propósito objeto de este trabajo. En mi opinión, este modelo es más adecuado para uso recreativo y con control manual, siendo las medidas de la IMU meramente informativas, puesto que es complicado realizar un control fino con ellas.

Sobre los algoritmos de control, se puede afirmar que son correctos, remitiéndonos a las pruebas. Sin embargo, su bondad es subjetiva respecto de las exigencias que les impongamos. Esto es, dependiendo de los requisitos de respuesta, se puede afirmar que son válidos o no para cierta aplicación.

En concreto, para la realización de las pruebas, se han elegido unos parámetros de control (primero por estimación y luego por tanteo) tal que al sistema le resulte asequible controlarse. Para el objeto de este trabajo, la consecución de trayectorias exitosas ya cubre los requerimientos de control. Máxime cuando usando un periodo de muestreo tan elevado, resulta inútil exigirle un buen rendimiento al control.

TRABAJOS FUTUROS

Los algoritmos de control desarrollados en este trabajo tienen una gran pega, y es que no manejan la dinámica real del sistema, sino que los parámetros de los reguladores son calculados por estimación y tanteo.

No es inusual tener que calibrar los reguladores, sin embargo, sería conveniente automatizar esta tarea. Un trabajo futuro podría ser la creación de algoritmos que en base a unas trayectorias de prueba, permitan caracterizar los reguladores por tanteo, igual que lo haría un humano, pero automatizando la tarea.

Un trabajo más fino sería crear algoritmos que caracterizaran en tiempo real la dinámica del sistema en base a las entradas y salidas del mismo. El problema es nuevamente el tiempo de muestre, ya que para estimar la dinámica del sistema harían falta un mínimo de muestras, y para cuando se hayan recopilado todas las muestras, la dinámica puede ser distinta por la posición y/o el rozamiento con el aire.

CAPÍTULO 7:
GESTIÓN DEL PROYECTO

- ***Planificación***
- ***Presupuesto***

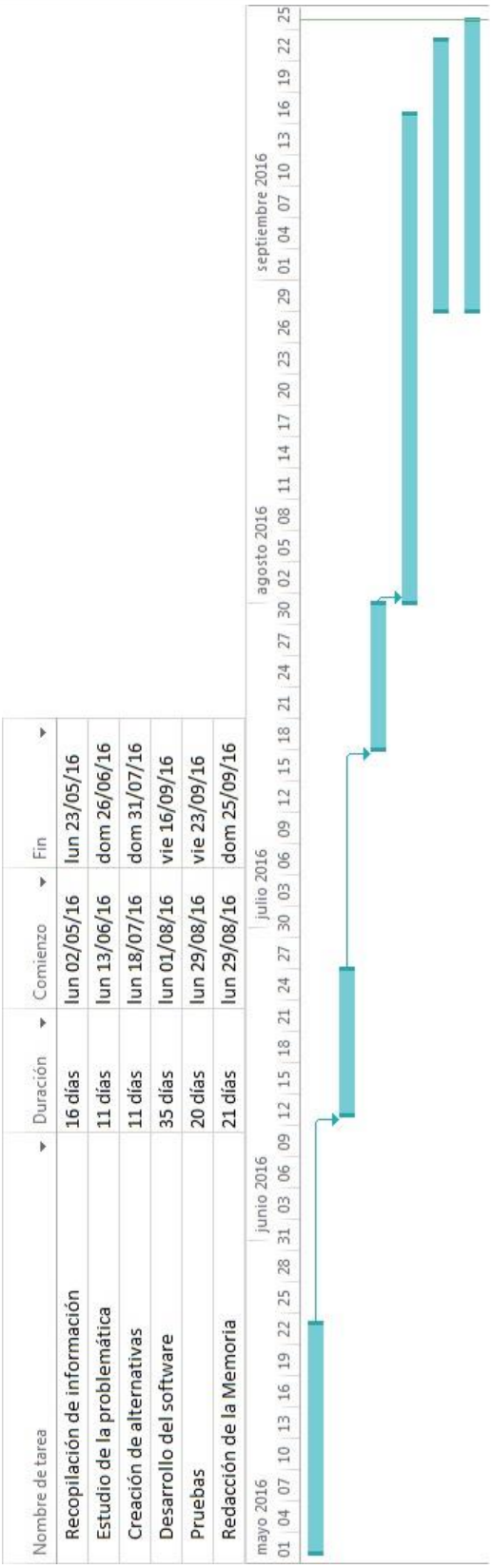
PLANIFICACIÓN

La planificación del trabajo se ilustra con un diagrama de Gantt, el cual es muy intuitivo, y rápidamente nos da idea de la inversión temporal.

Las tareas se han repartido en 6 puntos:

- Recopilación de información
- Estudio de la problemática
- Creación de alternativas
- Desarrollo del software
- Pruebas
- Redacción de la memoria

De los resultados mostrados en el siguiente gráfico, obtenemos el tiempo total invertido, que asciende a 114 días, lo que equivale a 912 horas, aplicando un cómputo de 8 horas diarias. Estos valores son solo estimaciones, puesto que al comienzo del proyecto el horario era más desatendido, y en su conclusión, más intensivo.



Nombre de tarea	Duración	Comienzo	Fin
Recopilación de información	16 días	lun 02/05/16	lun 23/05/16
Estudio de la problemática	11 días	lun 13/06/16	dom 26/06/16
Creación de alternativas	11 días	lun 18/07/16	dom 31/07/16
Desarrollo del software	35 días	lun 01/08/16	vie 16/09/16
Pruebas	20 días	lun 29/08/16	vie 23/09/16
Redacción de la Memoria	21 días	lun 29/08/16	dom 25/09/16

PRESUPUESTO

Para elaborar este presupuesto, se tiene en cuenta tanto los medios materiales invertidos en la elaboración del presente trabajo, como los medios humanos para su conclusión.

Para hacer el cómputo del coste humano, se establece un salario de ingeniero de 2000 € mensuales, 22 días laborales al mes, y 8 horas al día de jornada laboral, lo que computa 11'36€/h.

Capítulo 1. Medios materiales.

Código	Ud	Resumen	CanPres	PrPres	ImpPres
01.01	ud	Cuadricoptero marca Parrot, modelo ARDrone 2.0 Elite Edition Snow, completamente equipado	1	239,00 €	239,00 €
01.02	ud	Repuesto hélices para cuadricoptero marca Parrot, modelo ARDrone 2.0 Elite Edition.	1	7,01 €	7,01 €
01.03	ud	Repuesto engranajes para cuadricoptero marca Parrot, modelo ARDrone 2.0 Elite Edition.	1	9,99 €	9,99 €
01.04	ud	Repuesto motores para cuadricoptero marca Parrot, modelo ARDrone 2.0 Elite Edition.	2	39,00 €	78,00 €
01.05	ud	Kit herramienta extracción ejes de rotores Parrot, modelo ARDrone 2.0	1	13,00 €	13,00 €
01		**MEDIOS MATERIALES**			**347, 00 €**

Capítulo 2. Medios humanos.

Código	Ud	Resumen	CanPres	PrPres	ImpPres
02.01	h	Ingeniero de desarrollo junior	912	11,36 €	10360,32 €
02		**MEDIOS HUMANOS**			**10360,32 €**

Total

Código	Ud	Resumen	CanPres	PrPres	ImpPres
01		**MEDIOS MATERIALES**			**347, 00 €**
02		**MEDIOS HUMANOS**			**10360,32 €**
		TOTAL			**10707,32 €**

CAPÍTULO 8: DOCUMENTACIÓN DE CONSULTA

- ***Referencias***
- ***Bibliografía***

REFERENCIAS

[1] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Veh%C3%ADculo_a%C3%A9reo_no_tripulado.

[2] F. J. G. Muñoz, «Técnicas de control de vuelo sobre un cuadricóptero,» *Trabajo Fin de Grado,* p. 157, 2015.

[3] Parrot, «www.Parrot.com,» [En línea]. Available: https://www.parrot.com/es/drones/parrot-ardrone-20-elite-edition#parrotardrone-20-elite-edition.

[4] pixhawk.org, «pixhawk,» [En línea]. Available: https://pixhawk.org/.

[5] Microsoft, «msdn.microsoft.com,» [En línea]. Available: https://msdn.microsoft.com/es-es/library/ms171728(v=vs.110).aspx.

BIBLIOGRAFÍA

Garcia Muñoz, F. (2015). *Técnicas de control de vuelo sobre un cuadricóptero*. Leganes, Madrid: Universidad Carlos III.

Sharp, J. (2015). *Microsoft Visual C# Step by Step*. Redmond: Microsoft Press.

Microsoft. (s.f.). *Guía de programación de C#*. Obtenido de https://msdn.microsoft.com/es-es/library/67ef8sbd.aspx

Conde, J. (s.f.). Curso Visual C# 2012.

Piskorski, S., Brulez, N., Eline, P., & D'Haeyer, F. (2012). *Developer Guide SDK 2.0*. Parrot.

Ogata, K. (2010). *Ingeniería de Control Moderna*. Madrid: PEARSON EDUCACIÓN, S.A.

Moreno, L., Garrido, S., & Balaguer, C. (2003). *Ingeniería de Control. Modelado y control de sistemas dinámicos*. Ariel.

Colera Jimenez, J., García Pérez, R., & Oliveira Gonzalez, M. (2003). *Matemáticas II Bachillerato*. Barcelona: Grupo Anaya, S.A.

ANEXO A: DEFINICIONES DE TIPOS

- ***Librería de Clases***
- ***Formulario***

LIBRERÍA DE CLASES

Clase clsSystemInfo

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsSystemInfo.cs

Jerarquía de clases: ARDrone.FlightControl.clsSystemInfo

Sintaxis:

```
public class clsSystemInfo
```

Constructor:

```
public clsSystemInfo(DroneControl control, uint miliseconds,
uint free)
```

Atributos:

protected DroneData	_data	Datos de la IMU
protected DroneControl	_control	Control del drone
private bool	_linked	Atributo _control inicializado
private uint	_freedegrees	Grados de libertad
private System.Timers.Timer	tmrSample	Timer
private TimeSpan	period	Tiempo de muestreo
private DateTime	past	Instante anterior de muesteo
protected double[]	pose	Vector de posición
protected double[]	vel	Vector de velocidad
protected double	_speed	Módulo de velocidad lineal.
public HandlerControl	ControlEvent	Función delegada del control

Constantes:

```
public const uint __X = (uint)coord.X
public const uint __Y = (uint)coord.Y
public const uint __Z = (uint)coord.Z
public const uint __O = (uint)coord.O
```

Métodos:

private void	tmrSample_Tick(object sender, ElapsedEventArgs e)
	Evento del timer.
protected void	DoControl()

	Llama a la function delegada ControlEvent.
`protected virtual void`	`SystemStates()`
	Calcula los estados del Sistema. Posición y Velocidad.
`public virtual void`	`Reset()`
	Establece a cero la posición
`public virtual string`	`WriteInfo()`
	Devuelve una cadena con información de los estados.

Descriptores:

`public bool`	`Enable`
	Establece y devuelve la activación del control
`public DroneData`	`Data`
	Devuelve puntero a _data
`public DroneControl`	`Control`
	Devuelve puntero a _control
`public bool`	`Linked`
	Devuelve true si _control ha sido inicializado, y false en caso contrario.
`public uint`	`Free`
	Devuelve los grados de libertad.
`public TimeSpan`	`Period`
	Devuelve puntero a TimeSpan con el periodo de muestreo.
`public double[]`	`Pose`
	Devuelve puntero a vector de posición
`public double[]`	`Vel`
	Devuelve puntero a vector de velocidad
`public double`	`Speed`
	Devuelve módulo de velocidad lineal

Clase clsSystemInfo_IMU

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsSystemInfo.cs

Jerarquía de clases: ARDrone.FlightControl.clsSystemInfo

ARDrone.FlightControl.clsSystemInfo_IMU

Sintaxis:

```
public class clsSystemInfo_IMU : clsSystemInfo
```

Constructor:

```
public clsSystemInfo_IMU(DroneControl control, uint
miliseconds, uint free, ushort buf_vX = 1, ushort buf_vY
= 1, ushort buf_Z = 1, ushort buf_Pitch = 1, ushort
buf_Roll = 1, ushort buf_Yaw = 1)
: base(control, miliseconds, free)
```

Atributos:

public double[]	system_Vel	Velocida leida de la IMU
public double[]	system_Ang	Angulos leidos de la IMU
protected clsFilter	Filter_vX	Filtro para la velocidad en X
protected clsFilter	Filter_vY	Filtro para la velocidad en Y
protected clsFilter	Filter_Z	Filtro para la altura
protected clsFilter	Filter_Pitch	Filtro para ángulo de cabeceo
protected clsFilter	Filter_Roll	Filtro para ángulo de tonel
protected clsFilter	Filter_Yaw	Filtro para ángulo de guiñada
protected clsDifferentiator	Differentiator_sysVZ	Derivador de la altura
protected clsDifferentiator	Differentiator_sysVO	Derivador de la orientación
protected clsIntegrator	Integrator_X	Integrador de posición en X
protected clsIntegrator	Integrator_Y	Integrador de posición en Y
protected clsIntegrator	Integrator_Z	Integrador de posición en Z
protected clsIntegrator	Integrator_O	Integrador de la orientación

Métodos:

protected override void	SystemStates()
	Calcula los estados del Sistema. Posición y Velocidad.
public override void	Reset()
	Establece a cero la posición

`public override string WriteInfo()`	Devuelve una cadena con información de los estados.

Clase clsSystemInfo_IMAGE

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsSystemInfo.cs

Jerarquía de clases: ARDrone.FlightControl.clsSystemInfo

ARDrone.FlightControl.clsSystemInfo_IMAGE

Sintaxis:

```
public class clsSystemInfo_IMAGE : clsSystemInfo
```

Constructor:

```
public clsSystemInfo_IMAGE(DroneControl control, uint
miliseconds, uint free, double[] pData)
 : base(control, miliseconds, free)
```

Atributos:

`public double[]`	`_imagedata`	Datos
`protected clsDifferentiator`	`Differentiator_VX`	Derivador de velocidad en X
`protected clsDifferentiator`	`Differentiator_VY`	Derivador de velocidad en Y
`protected clsDifferentiator`	`Differentiator_VZ`	Derivador de velocidad en Z
`protected clsDifferentiator`	`Differentiator_VO`	Derivador de la orientación
`protected clsIntegrator`	`Integrator_X`	Integrador de posición en X
`protected clsIntegrator`	`Integrator_Y`	Integrador de posición en Y
`protected clsIntegrator`	`Integrator_Z`	Integrador de posición en Z
`protected clsIntegrator`	`Integrator_O`	Integrador de la orientación

Métodos:

`protected override void`	`SystemStates()`
	Calcula los estados del Sistema. Posición y Velocidad.
`public override void`	`Reset()`
	Establece a cero la posición
`public override string`	`WriteInfo()`
	Devuelve una cadena con información de los estados.

Clase clsControlSystem

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsControlSystem.cs

Jerarquía de clases: ARDrone.FlightControl.clsControlSystem

Sintaxis:

```
public class clsControlSystem
```

Constructor:

```
public clsControlSystem(DroneControl control, uint miliseconds, uint free, DataType type, double[] imagedata = null)
```

Atributos:

public clsSystemInfo	_system	Estados del sistema
private StreamWriter	file	Archivo de grabación de telemetría
protected bool	_recording	Estado de activación de la grbación
protected double[]	error_pose	Error de posición
protected double[]	error_vel	Error de velocidad
protected double[]	target_pose	Posición de consigna
protected double[]	target_vel	Velocidad de consigna
protected double[]	target_act	Acción de consgna
protected uint	max_vel_lin	Velocidad lineal máxima
protected double	max_vel_ang	Velocidad angular máxima
protected float	cmdPitch	Valor comando de Pitch
protected float	cmdRoll	Valor comando de Roll
protected float	cmdYaw	Valor comando de Yaw
protected float	cmdGaz	Valor comando de Gaz
protected clsP_I_D_Regulator	Regulator_X	Regulador para la posición eje X
protected clsP_I_D_Regulator	Regulator_Y	Regulador para la posición eje Y
protected clsP_I_D_Regulator	Regulator_Z	Regulador para la posición eje Z
protected clsP_I_D_Regulator	Regulator_O	Regulador para la posición eje O
protected clsP_I_D_Regulator	Regulator_VX	Regulador para la velocidad eje X
protected clsP_I_D_Regulator	Regulator_VY	Regulador para la velocidad eje Y
protected clsP_I_D_Regulator	Regulator_VZ	Regulador para la velocidad eje Z
protected clsP_I_D_Regulator	Regulator_VO	Regulador para la velocidad eje O

Constantes:

```
public const uint   __X = (uint)coord.X
public const uint   __Y = (uint)coord.Y
public const uint   __Z = (uint)coord.Z
public const uint   __O = (uint)coord.O
```

Métodos:

```
protected virtual void   Control_Action()
```
Realiza la acción de control

```
protected void   Error_Pose()
```
Calcula el error de posición

```
protected void   Error_Vel()
```
Calcula el error de velocidad

```
protected void   Regulator_Pose()
```
Regulador de posición. Calcula la velocidad de consigna.

```
protected void   Regulator_Vel()
```
Regulador de velocidad. Calcula la acción de consigna de los actuadores

```
protected void   Limit_Vel()
```
Limita la velocidad de consigna

```
protected void   Actuators()
```
Ordena al drone el movimiento de control

```
public void   Reset()
```
Reinicia los reguladores y el origen de coordenadas

```
public virtual void   Static(double X = 0, double Y = 0, double Z = 0, double O = 0)
```
Introduce un destino para el control estático

```
protected void   SaveData()
```
Guarda la telemetría.

```
public void   Start()
```
Inicia el timer de muestreo

```
public void   Stop()
```
Para el timer de muestreo

```
public void   ActiveControl(bool active)
```
Activa el control inicializando el delegado del evento de control del objeto _system

```
public bool   ActiveControl()
```
Devuelve el estado del delegado del evento de control del objeto _system

```
public void   StartRecord(string s)
```
Inicia la grabación de la telemetría

`public void`	`StopRecord()` Detiene la grabación de la telemetría

Descriptores:

`public bool`	`Enable` Establece y devuelve el estado de activación del timer de muestreo
`public bool`	`Recording` Devuelve true si está activa la de grabación de telemetría
`public uint`	`Period` Devuelve el periodo de muestreo
`public double[]`	`Pose` Devuelve un puntero al vector posición
`public double[]`	`Vel` Devuelve un puntero al vector velocidad
`public double`	`Speed` Devuelve el valor de velocidad lineal absoluta
`public bool`	`Linked` Devuelve el valor de _system.Linked
`public uint`	`Max_V_Lineal` Establece y devuelve la velocidad lineal máxima
`public double`	`Max_V_Angular` Establece y devuelve la angular lineal máxima
`public double`	`RegX_P` Establece y devuelve el parámetro P del regulador de posición en el eje X
`public double`	`RegX_I` Establece y devuelve el parámetro I del regulador de posición en el eje X
`public double`	`RegX_D` Establece y devuelve el parámetro D del regulador de posición en el eje X
`public double`	`RegY_P` Establece y devuelve el parámetro P del regulador de posición en el eje Y
`public double`	`RegY_I` Establece y devuelve el parámetro I del regulador de posición en el eje Y
`public double`	`RegY_D` Establece y devuelve el parámetro D del regulador de posición en el eje Y
`public double`	`RegZ_P` Establece y devuelve el parámetro P del regulador de posición en el eje Z

`public double` `RegZ_I`
Establece y devuelve el parámetro I del regulador de posición en el eje Z

`public double` `RegZ_D`
Establece y devuelve el parámetro D del regulador de posición en el eje Z

`public double` `RegO_P`
Establece y devuelve el parámetro P del regulador de posición en el eje O

`public double` `RegO_I`
Establece y devuelve el parámetro I del regulador de posición en el eje O

`public double` `RegO_D`
Establece y devuelve el parámetro D del regulador de posición en el eje O

`public double` `RegVX_P`
Establece y devuelve el parámetro P del regulador de velocidad en el eje X

`public double` `RegVX_I`
Establece y devuelve el parámetro I del regulador de velocidad en el eje X

`public double` `RegVX_D`
Establece y devuelve el parámetro D del regulador de velocidad en el eje X

`public double` `RegVY_P`
Establece y devuelve el parámetro P del regulador de velocidad en el eje Y

`public double` `RegVY_I`
Establece y devuelve el parámetro I del regulador de velocidad en el eje Y

`public double` `RegVY_D`
Establece y devuelve el parámetro D del regulador de velocidad en el eje Y

`public double` `RegVZ_P`
Establece y devuelve el parámetro P del regulador de velocidad en el eje Z

`public double` `RegVZ_I`
Establece y devuelve el parámetro I del regulador de velocidad en el eje Z

`public double` `RegVZ_D`
Establece y devuelve el parámetro D del regulador de velocidad en el eje Z

`public double`	`RegVO_P` Establece y devuelve el parámetro P del regulador de velocidad en el eje O
`public double`	`RegVO_I` Establece y devuelve el parámetro I del regulador de velocidad en el eje O
`public double`	`RegVO_D` Establece y devuelve el parámetro D del regulador de velocidad en el eje O
`public double[]`	`Target_Pose` Devuelve un puntero al vector de posición de consigna
`public double[]`	`Target_Vel` Devuelve un puntero al vector de velocidad de consigna
`public double[]`	`Target_Act` Devuelve un puntero al vector de acción de consigna
`public double`	`CMD_Pitch` Devuelve el valor de la orden de Pitch al drone
`public double`	`CMD_Roll` Devuelve el valor de la orden de Roll al drone
`public double`	`CMD_Yaw` Devuelve el valor de la orden de Yaw al drone
`public double`	`CMD_Gaz` Devuelve el valor de la orden de Gaz al drone

Clase clsNavigationSystem

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsNavigationSystem.cs

Jerarquía de clases: ARDrone.FlightControl.clsControlSystem

ARDrone.FlightControl.NavigationSystem

Sintaxis:

```
public class clsNavigationSystem : clsControlSystem
```

Constructor:

```
public clsNavigationSystem(DroneControl control, uint
miliseconds, uint free, DataType type, double[]
imagedata = null ,double margin = 250)
: base(control, miliseconds, free, type, imagedata)
```

Atributos:

`private double[]`	`_origin`	Origen de la trayectoria
`private double[]`	`_destiny`	Destino de la trayectoria
`private double`	`_progress`	Progreso en tanto por uno
`private double`	`_distanceToOrigin`	Distancia recorrida
`private double`	`_distanceToDestiny`	Distancia restante
`private bool`	`stay_origin`	Situación en origen
`private bool`	`stay_destiny`	Situación en destino
`private bool`	`_static`	Estado de control estático
`private double`	`_margin`	Margen de consideración de destino
`public event`	`EventHandler Arrive`	Evento de llegada a destino

Métodos:

`protected virtual void`	`OnArrive()` Llamada a evento de llegada
`protected override void`	`Control_Action()` Realiza la acción de control
`private void`	`Trajectory_Info()` Calcula los parámetros de la trayectoria
`private void`	`CruiseSpeed()` Calcula la velocidad de navegación

`public void`	`ActiveControl(ControlType type)` Activa el control inicializando el delegado del evento de control del objeto heredado _system
`public override void`	`Static(double X = 0, double Y = 0, double Z = 0, double O = 0)` Introduce un destino para el control estático
`public void`	`NewDestiny(double X = 0, double Y = 0, double Z = 0, double O = 0)` Introduce un destino para el control dinámico
`public void`	`Destiny(double X = 0, double Y = 0, double Z = 0, double O = 0)` Establece el vector de destino
`public void`	`Origin(double X = 0, double Y = 0, double Z = 0, double O = 0)` Establece el vector de origen
`public void`	`ReadyToNewDestiny()` Prepara la clase para la introducción de destinos para el control dinámico
`public double`	`Progress()` Devuelve el progreso de la trayectoria

Descriptores:

`public double`	`DistanceToOrigin` Devuelve la distancia recorrida por el drone en la trayectoria
`public double`	`DistanceToDestiny` Devuelve la distancia restante de la trayectoria
`public bool`	`IsOnOrigin` Devuelve true si el drone se encuentra en el origen
`public bool`	`IsOnDestiny` Devuelve true si el drone se encuentra ha llegado a su destino

Clase clsCommands

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsCommands.cs

Jerarquía de clases: ARDrone.FlightControl.clsCommands

Sintaxis:

```
public class clsCommands
```

Constructor:

`public`	`clsCommands(DroneControl control)`

Atributos:

`private DroneControl`	_control	Control del drone

Métodos:

`public string`	`Takeoff()` Ejecuta el despegue
`public string`	`Land()` Ejecuta el despegue
`public string`	`Emergency()` Activa y reestablece la señal de emergencia
`public string`	`FlatTrim()` Reajusta los giróscopos devolviendo a cero la horizontalidad
`public string`	`EnterHoverMode()` Entrar en estado de suspensión en el aire
`public string`	`LeaveHoverMode()` Salir del estado de suspensión en el aire
`public void`	`Navigate(float roll, float pitch, float yaw, float gaz)` Ejecutar comandos de movimiento

Clase clsFlightPlan

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsFlightPlan.cs

Jerarquía de clases: ARDrone.FlightControl.clsFlightPlan

Sintaxis:

```
public class clsFlightPLan
```

Constructor:

```
public   clsFlightPlan(CheckedListBox screen =
         (CheckedListBox)null)
```

Atributos:

```
private List<clsPoint>   _Points
private int              _Index
private CheckedListBox   _Screen
```

Métodos:

`public void` `connectTo(CheckedListBox screen)`
Vincular la clase con un control de tipo *CheckedListBox*

`public clsPoint` `FlyAway()`
Devuelve la primera trayectoria de la lista de trayectorias

`public clsPoint` `NextPoint()`
Devuelve el siguiente punto en la lista de trayectorias

`public void` `AddPoint(double X, double Y, double Z, double O)`
Añade un destino a la lista

`public void` `DeletePoint()`
Borra un destino de la lista

`public void` `ClearPoints()`
Borra todos los puntos de la lista

`public void` `SavePoints(string file)`
Salva la lista en un archive de texto plano

`public void` `LoadPoints(string file)`
Carga una lista de un archivo de texto plano

`private void` `RefreshScreen()`
Actualiza el control *CheckedListBox*

Clase clsAutomaticPilot

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsAutomaticPilot.cs

Jerarquía de clases: ARDrone.FlightControl.clsCommands

ARDrone.FlightControl.clsAutomaticPilot

Sintaxis:

```
public class clsAutomaticPilot : clsCommands
```

Constructor:

```
public   clsAutomaticPilot(DroneControl control, DataType type,
         double[] imagedata = null)
         : base(control)
```

Atributos:

```
   private DroneControl   _control
private clsControlSystem  _flightControl
  private clsFlightPlan   _plan
public event EventHandler<StringArgs> Report
```

Métodos:

```
           public void   connectToView(CheckedListBox screen)

 protected virtual void  OnReport(StringArgs e)

           public void   Fly()
          private void   _flightControl_Arrive(object sender, EventArgs e)

           public void   Static(double X, double Y, double Z, double O)

          private void   MustBeFlying()

           public void   Reset()

           public void   Add(double X, double Y, double Z, double O)

           public void   Delete()

           public void   Clear()
```

`public void` `Save(string s)`

`public void` `Load(string s)`

`public void` `Start()`
Inicia el timer de muestreo

`public void` `Stop()`
Para el timer de muestreo

`public void` `StartRecord(string s)`
Inicia la grabación de la telemetría

`public void` `StopRecord()`
Detiene la grabación de la telemetría

Delegados:

`public bool` `Enable`
Establece y devuelve el estado de activación del timer de muestreo

`public bool` `ActiveControl`
Devuelve el estado del control

`public bool` `Recording`
Devuelve true si está activa la de grabación de telemetría

`public DroneControl` `Control`
Devuelve un puntero a la clase _control

`public uint` `Period`
Devuelve el periodo de muestreo

`public double` X
Devuelve la posición del eje X

`public double` Y
Devuelve la posición del eje Y

`public double` Z
Devuelve la posición del eje Z

`public double` O
Devuelve la posición del eje O

`public double` vX
Devuelve la velocidad del eje X

`public double` vY
Devuelve la velocidad del eje Y

`public double` vZ
Devuelve la velocidad del eje Z

`public double` vO
Devuelve la velocidad del eje O

`public double` `Speed`
Devuelve el valor de velocidad lineal absoluta

`public bool` `Linked`
Devuelve el valor de _flightcontrol.Linked

`public double` `Progress()`
Devuelve el progreso de la trayectoria

`public uint` `LimitLinealVelocity`
Establece y devuelve el límite de velocidad lineal

`public double` `LimitAngularVelocity`
Establece y devuelve el límite de velocidad angular

`public double` `Reg_XY_P`
Establece y devuelve el parámetro P del regulador de posición en los ejes X e Y

`public double` `Reg_Z_P`
Establece y devuelve el parámetro P del regulador de posición en el eje Z

`public double` `Reg_O_P`
Establece y devuelve el parámetro P del regulador de posición en el eje O

`public double` `Reg_vXY_P`
Establece y devuelve el parámetro P del regulador de velocidad en los ejes X e Y

`public double` `Reg_vZ_P`
Establece y devuelve el parámetro P del regulador de velocidad en el eje Z

`public double` `Reg_vO_P`
Establece y devuelve el parámetro P del regulador de velocidad en el eje O

`public double` `Reg_vXY_I`
Establece y devuelve el parámetro I del regulador de velocidad en los ejes X e Y

`public double` `Reg_vZ_I`
Establece y devuelve el parámetro I del regulador de velocidad en el eje Z

`public double` `Reg_vO_I`
Establece y devuelve el parámetro I del regulador de velocidad en el eje O

`public double` `cmdPitch`
Devuelve el valor de la orden de Pitch al drone

`public double` `cmdRoll`
Devuelve el valor de la orden de Roll al drone

`public double` `cmdYaw`
Devuelve el valor de la orden de Yaw al drone

`public double` `cmdGaz`
Devuelve el valor de la orden de Gaz al drone

Clase clsP_I_D_Regulator

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsP_I_D_Regulator.cs

Jerarquía de clases: ARDrone.FlightControl.clsP_I_D_Regulator

Sintaxis:

```
public class clsP_I_D_Regulator
```

Constructor:

```
public   clsP_I_D_Regulator(double P = 0, double I = 0, double
         D = 0)
```

Atributos:

`private double`	`_action`	Valor de control
`private clsIntegrator`	`error_acum`	Integrados
`private clsDifferentiator`	`error_diff`	Diferenciador
`private double`	`Kp, Ki, Kd`	Parámetros del regulador

Métodos:

`public double` `Sample(double error, TimeSpan period)`
Introducir una muestra en el regulador

`public void` `Reset()`
Reiniciar regulador

`public void` `Change(double P = 0, double I = 0, double D = 0)`
Cambiar parámetros

Descriptores:

`public double` `Value`
Devuelve el valor de control

`public double` `P`
Establece y devuelve el parámetro proporcional

`public double` `I`
Establece y devuelve el parámetro integral

`public double` `D`
Establece y devuelve el parámetro diferencial

Clase clsDifferentiator

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsDifferentiator.cs

Jerarquía de clases: ARDrone.FlightControl.clsDifferentiator

Sintaxis:

```
public class clsDifferentiator
```

Constructor:

`public`	`clsDifferentiator()`

Atributos:

`protected double`	`Difference`
`protected double`	`a0`
`protected DateTime`	`past`

Métodos:

`public virtual double`	`Sample(double a1)` Introducir una muestra en el diferenciador
`public virtual double`	`Sample(double a1, TimeSpan time)` Introducir una muestra en el diferenciador

Descriptores:

`public double`	`Value` Establece y devuelve el valor de la operación

Clase clsDifferentiatorAngle

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsDifferentiator.cs

Jerarquía de clases: ARDrone.FlightControl.clsDifferentiator

ARDrone.FlightControl.clsDifferentiatorAngle

Sintaxis:

```
public class clsDifferentiatorAngle : clsDifferentiator
```

Constructor:

```
public   clsDifferentiatorAngle()
```

Métodos:

`public override double`	`Sample(double a1)` Introducir una muestra en el diferenciador
`public override double`	`Sample(double a1, TimeSpan time)` Introducir una muestra en el diferenciador

Clase clsIntegrator

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsIntegrator.cs

Jerarquía de clases: ARDrone.FlightControl.clsIntegrator

Sintaxis:

```
public class clsIntegrator
```

Constructor:

```
public   clsIntegrator()
```

Atributos:

```
protected double   Integral
protected double   a0
protected DateTime   past
```

Métodos:

```
public virtual double   Sample(double a1)
```
Introducir una muestra en el integrador

```
public virtual double   Sample(double a1, TimeSpan time)
```
Introducir una muestra en el integrador

```
public void   Reset()
```
Reinicia el valor acumulado estableciéndolo de nuevo en cero

Descriptores:

```
public double   Value
```
Establece y devuelve el valor de la operación

Clase clsIntegratorAngle

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsIntegrator.cs

Jerarquía de clases: ARDrone.FlightControl.clsIntegrator

ARDrone.FlightControl.clsIntegratorAngle

Sintaxis:

```
public class clsIntegratorAngle : clsIntegrator
```

Constructor:

```
public   clsIntegratorAngle()
```

Métodos:

```
public virtual double   Sample(double a1)
```
Introducir una muestra en el integrador

```
public virtual double   Sample(double a1, TimeSpan time)
```
Introducir una muestra en el integrador

Clase clsFilter

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsFilter.cs

Jerarquía de clases: ARDrone.FlightControl.clsFilter

Sintaxis:

```
public class clsFilter
```

Constructor:

```
public   clsFilter(ushort n)
```

Atributos:

```
private double   _mean
private double[] _buffer
private ushort   _numSamples
private ushort   _index
```

Métodos:

```
public double   Sample(double s)
```
Introducir una muestra en el filtro

```
public void   Reset()
```
Reinicia filtro estableciendo todos su valores de nuevo en cero

Descriptores:

```
public double   Value
```
Devuelve el valor filtrado

Clase clsPoint

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsFlightPlan.cs

Jerarquía de clases: ARDrone.FlightControl.clsPoint

Sintaxis:

```
public class clsPoint
```

Constructor:

```
public   clsPoint()
```

Atributos:

`public double[]`	`Point`	Vector punto
`public status`	`Status`	`Estado del punto`

Clase StringArgs

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsAutomaticPilot.cs

Jerarquía de clases: System.Object

System.EventArgs

ARDrone.FlightControl.StringArgs

Sintaxis:

```
public class StringArgs : EventArgs
```

Descriptores:

`public string` `str`

Establece y devuelve una cadena de tipo string

Delegados

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsSystemInfo.cs

```
public delegate void   HandlerControl()
```

Tipos enumerados

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsSystemInfo.cs

```
public enum   coord : uint { X = 0, Y, Z, O }
public enum   angles : uint { Pitch = 0, Roll, Yaw }
```

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsControlSystem.cs

```
public enum   DataType : uint { IMU = 0, IMAGE, GPS }
```

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsNavigationSystem.cs

```
public enum   ControlType : uint {NO = 0, STATIC, DINAMIC}
```

Espacio de Nombres: ARDrone.FlightControl

Archivo: clsFlightPlan.cs

```
public enum   status : uint { EMPTY = 0, GOOD, BAD, END }
```

FORMULARIO

Controles

```
private System.Windows.Forms.GroupBox grpStatus
private System.Windows.Forms.TrackBar trk_Gaz
private System.Windows.Forms.Label lbl_Pitch
private System.Windows.Forms.TrackBar trk_Pitch
private System.Windows.Forms.Label lbl_Roll
private System.Windows.Forms.TrackBar trk_Roll
private System.Windows.Forms.Label lbl_Yaw
private System.Windows.Forms.TrackBar trk_Yaw
private System.Windows.Forms.Label lbl_Gaz
private System.Windows.Forms.Label lblPeriod
private System.Windows.Forms.Label lblSpeed
private System.Windows.Forms.Label lbl_vO
private System.Windows.Forms.Label lbl_O
private System.Windows.Forms.Label lbl_vZ
private System.Windows.Forms.Label lbl_vY
private System.Windows.Forms.Label lbl_vX
private System.Windows.Forms.Label lbl_Z
private System.Windows.Forms.Label lbl_Y
private System.Windows.Forms.Label lbl_X
private System.Windows.Forms.GroupBox grpParameter
private System.Windows.Forms.GroupBox grpVelocity
private System.Windows.Forms.Label lblI_vO
private System.Windows.Forms.TextBox txtI_vO
private System.Windows.Forms.Label lblI_vZ
private System.Windows.Forms.TextBox txtI_vZ
private System.Windows.Forms.Label lblI_vXY
private System.Windows.Forms.TextBox txtI_vXY
private System.Windows.Forms.Label lblP_vO
private System.Windows.Forms.TextBox txtP_vO
private System.Windows.Forms.Label lblP_vZ
private System.Windows.Forms.TextBox txtP_vZ
private System.Windows.Forms.Label lblP_vXY
private System.Windows.Forms.TextBox txtP_vXY
```

```
private System.Windows.Forms.GroupBox grpLineal
private System.Windows.Forms.Label lblP_O
private System.Windows.Forms.TextBox txtP_O
private System.Windows.Forms.Label lblP_Z
private System.Windows.Forms.TextBox txtP_Z
private System.Windows.Forms.Label lblP_XY
private System.Windows.Forms.TextBox txtP_XY
private System.Windows.Forms.GroupBox grpLimits
private System.Windows.Forms.Label lblLimitAng
private System.Windows.Forms.TextBox txtMax_VAng
private System.Windows.Forms.Label lblLimitLin
private System.Windows.Forms.TextBox txtMax_VLin
private System.Windows.Forms.Button btnSet
private System.Windows.Forms.GroupBox grpCoordinates
private System.Windows.Forms.Button btnSTATIC
private System.Windows.Forms.Label lblO
private System.Windows.Forms.TextBox txtO
private System.Windows.Forms.Label lblZ
private System.Windows.Forms.TextBox txtZ
private System.Windows.Forms.Label lblY
private System.Windows.Forms.TextBox txtY
private System.Windows.Forms.Label lblX
private System.Windows.Forms.TextBox txtX
private System.Windows.Forms.GroupBox grpTrajectory
private System.Windows.Forms.PictureBox picLink
private System.Windows.Forms.Button btnReset
private System.Windows.Forms.Button btnLoad
private System.Windows.Forms.Button btnSave
private System.Windows.Forms.Button btnClear
private System.Windows.Forms.Button btnDelete
private System.Windows.Forms.Button btnTakeOff
private System.Windows.Forms.Button btnFly
private System.Windows.Forms.Button btnSTART_STOP
private System.Windows.Forms.Button btnAdd
private System.Windows.Forms.Label lblStatus
private System.Windows.Forms.ProgressBar pgbProgress
private System.Windows.Forms.Button btnRECORD
```

```
private System.Windows.Forms.CheckedListBox lstTrajectory
```

Clase FlightControlForm

Espacio de Nombres: LSIDroneInterface

Archivo: FlightControlForm.cs

Jerarquía de clases: . . .

System.Windows.Forms.Form

LSIDroneInterface.FlightControlForm

Sintaxis:

```
public class FlightControlForm : Form
```

Constructor:

```
public FlightControlForm(MainForm _Parent, DroneControl
_Control, DataType type, double[] imagedata = null)
```

Atributos:

```
private new MainForm Parent
private clsAutomaticPilot Otto
private System.Timers.Timer tmrUpdateGUI
private string StatusText
private String _control_label
private System.Drawing.Color _control_color
private readonly System.Drawing.Color _original_color =
  System.Drawing.SystemColors.Control
private readonly System.Drawing.Color _green_color =
  System.Drawing.Color.FromArgb(((int)(((byte)(192)))),
  ((int)(((byte)(255)))), ((int)(((byte)(192)))))
private readonly System.Drawing.Color _red_color =
  System.Drawing.Color.FromArgb(((int)(((byte)(255)))),
  ((int)(((byte)(128)))), ((int)(((byte)(128)))))
private readonly System.Drawing.Color _redred_color = System.Drawing.Color.Red
```

Métodos:

```
public void Otto_Report(object sender, StringArgs e)
```
Mostrar en texto en control *lblStatus*

```
private void InitializeGUITimer()
```
Inicializar el timer de refresco del GUI

```
private void   tmrUpdateGUI_Tick(object  sender,  ElapsedEventArgs  e)
```

EVento del timer. Llama a "UpdateGUI()"

```
private void   UpdateGUI()
```

Actualiza los controles del formulario

Eventos de controles:

```
private void   FlightControlForm_Load(object sender, EventArgs e)
private void   FlightControlForm_Close(object                sender,
               FormClosingEventArgs e)
private void   FlightControlForm_KeyPress(object             sender,
               KeyPressEventArgs e)
private void   btnSet_Click(object sender, EventArgs e)
private void   btnRECORD_Click(object sender, EventArgs e)
private void   btnAdd_Click(object sender, EventArgs e)
private void   btnDelete_Click(object sender, EventArgs e)
private void   btnClear_Click(object sender, EventArgs e)
private void   btnSave_Click(object sender, EventArgs e)
private void   btnLoad_Click(object sender, EventArgs e)
private void   txtP_XY_TextChanged(object sender, EventArgs e)
private void   txtP_Z_TextChanged(object sender, EventArgs e)
private void   txtP_O_TextChanged(object sender, EventArgs e)
private void   txtP_vXY_TextChanged(object sender, EventArgs e)
private void   txtP_vZ_TextChanged(object sender, EventArgs e)
private void   txtP_vO_TextChanged(object sender, EventArgs e)
private void   txtI_vXY_TextChanged(object sender, EventArgs e)
private void   txtI_vZ_TextChanged(object sender, EventArgs e)
private void   txtI_vO_TextChanged(object sender, EventArgs e)
private void   txtMax_VLin_TextChanged(object sender, EventArgs e)
private void   txtMax_VAng_TextChanged(object sender, EventArgs e)
private void   txtX_TextChanged(object sender, EventArgs e)
private void   txtY_TextChanged(object sender, EventArgs e)
private void   txtZ_TextChanged(object sender, EventArgs e)
private void   txtO_TextChanged(object sender, EventArgs e)
```

Llamadas a controles en hilo seguro:

```
private delegate void SetTextCallback(String text)
private delegate void SetTrackCallback(int value, System.Drawing.Color color)
private delegate void SetBarCallback(int value)
private delegate void SetButtonCallback(String text, System.Drawing.Color color)
```

```
private delegate void SetPictureCallback(System.Drawing.Color color)
            private void   SetTextStatus(String text)
            private void   SetPictureLink(System.Drawing.Color color)
            private void   SetTextPeriod(String text)
            private void   SetTextPosX(String text)
            private void   SetTextPosY(String text)
            private void   SetTextPosZ(String text)
            private void   SetTextPosO(String text)
            private void   SetTextVelX(String text)
            private void   SetTextVelY(String text)
            private void   SetTextVelZ(String text)
            private void   SetTextVelO(String text)
            private void   SetTextSpeed(String text)
            private void   SetTextPitch(String text)
            private void   SetTextRoll(String text)
            private void   SetTextYaw(String text)
            private void   SetTextGaz(String text)
            private void   SetTrackPitch(int value, System.Drawing.Color color)
            private void   SetTrackRoll(int value, System.Drawing.Color color)
            private void   SetTrackYaw(int value, System.Drawing.Color color)
            private void   SetTrackGaz(int value, System.Drawing.Color color)
            private void   SetBarProgress(int value)
            private void   SetButtonSTART_STOP(String  text,  System.Drawing.Color
                           color)
            private void   SetButtonTakeOff(String   text,   System.Drawing.Color
                           color)
            private void   SetButtonRECORD(String    text,    System.Drawing.Color
                           color)
```

ANEXO B:
CÁLCULOS DE NAVEGACIÓN

- ***Cálculo de la trayectoria***
- ***Cálculo de la velocidad***

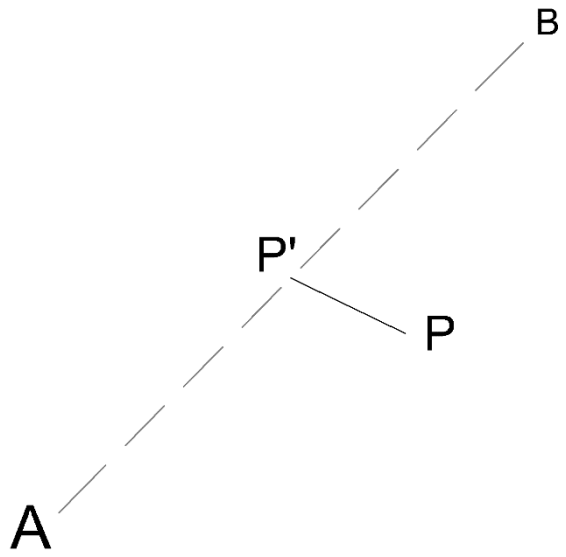
CÁLCULO DE LA TRAYECTORIA

Para poder controlar al drone en torno a una trayectoria, es necesario conocer la posición que debería ocupar el drone en la trayectoria, para compararla con la posición real y obtener así el error de posición que deberá ser ajustado por medio del control.

Se ha considerado al punto de la trayectoria más cercano al drone, como el punto que este debería ocupar.

Supongamos que el drone se desplaza desde el punto A (origen), hasta el punto B (destino), siendo el punto P la posición actual del drone, el punto P', su posición ideal. El vector P'P representaría el error de posición.

Para obtener P', sabemos que los vectores AB y P'P han de ser perpendiculares, y por lo tanto, su producto escalar ha de ser igual a cero.



$$\boldsymbol{\overrightarrow{AB} \cdot \overrightarrow{P'P} = 0}$$

$$\left.\begin{matrix}\overrightarrow{P'P} = \vec{P} - \overrightarrow{P'} \\ \overrightarrow{P'} = \vec{A} + k\overrightarrow{AB}\end{matrix}\right\} \overrightarrow{P'P} = \vec{P} - \left(\vec{A} + k\overrightarrow{AB}\right)$$

$$\overrightarrow{P'P} = \overrightarrow{AP} - k\overrightarrow{AB}$$
$$\overrightarrow{AB} = \vec{B} - \vec{A}$$

$$\overrightarrow{AB} \cdot \overrightarrow{P'P} = 0$$

$$AB_X \cdot AP_X - kAB_X{}^2 + AB_Y \cdot AP_Y - kAB_Y{}^2 + AB_Z \cdot AP_Z - kAB_Z{}^2 = 0$$

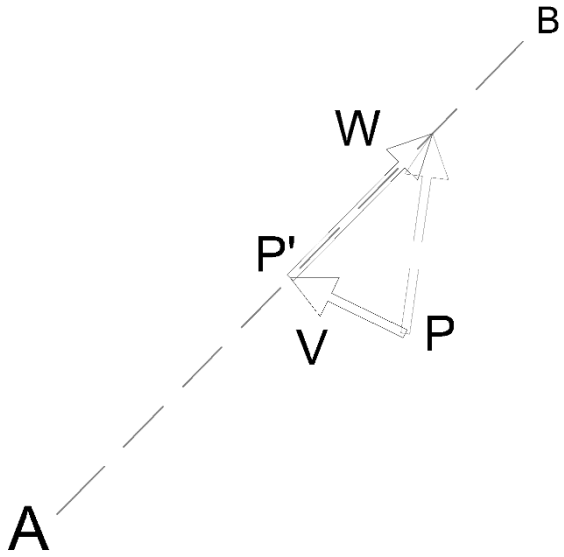
$$\boldsymbol{k = \frac{AB_X \cdot AP_X + AB_Y \cdot AP_Y + AB_Z \cdot AP_Z}{kAB_X{}^2 + kAB_Y{}^2 + kAB_Z{}^2}}, \qquad \boldsymbol{\overrightarrow{P'} = \vec{A} + k\overrightarrow{AB}}$$

CÁLCULO DE LA VELOCIDAD

En el algoritmo de control dinámico, la posición de consigna no es el destino, sino un punto de la trayectoria. Esto provoca que el control tienda a mantener al drone en la trayectoria, olvidándose de avanzar en ella.

Para forzar al drone a avanzar en la trayectoria, a la velocidad de corrección de trayectoria (V), se le ha de sumar un vector velocidad con la misma directriz que la trayectoria (W).

Para que el cálculo sea optimo, el módulo de la suma de ambos vectores debe ser igual a la velocidad máxima indicada por el usuario en la interfaz gráfica.



$$\left|\vec{V} + \vec{W}\right| = \boldsymbol{V\_max}$$

$$\vec{W} = k\vec{AB}$$

$$(V_X + kAB_X)^2 + (V_Y + kAB_Y)^2 + (V_Z + kAB_Z)^2 = V\_max^2$$

$$V_X{}^2 + AB_X{}^2 \cdot k^2 + 2V_XAB_X \cdot k + V_Y{}^2 + AB_Y{}^2 \cdot k^2 + 2V_YAB_Y \cdot k + V_Z{}^2 + AB_Z{}^2 \cdot k^2 + 2V_ZAB_Z \cdot k - V_{max}{}^2 = 0$$

$$\left(AB_X{}^2 + AB_Y{}^2 + AB_Z{}^2\right) \cdot k^2 + 2(V_XAB_X + V_YAB_Y + V_ZAB_Z) \cdot k + \left(V_X{}^2 + V_Y{}^2 + V_Z{}^2 - V_{max}{}^2\right) = 0$$

$$\left.\begin{array}{c} \boldsymbol{a = \left(AB_X{}^2 + AB_Y{}^2 + AB_Z{}^2\right)} \\ \boldsymbol{b = 2(V_XAB_X + V_YAB_Y + V_ZAB_Z)} \\ \boldsymbol{c = \left(V_X{}^2 + V_Y{}^2 + V_Z{}^2 - V_{max}{}^2\right)} \end{array}\right\} \rightarrow \boldsymbol{k = \frac{-b + \sqrt{b^2 - 4ac}}{2a}}$$

$$\vec{W} = k\vec{AB}$$